# Magnetic Field Visualization for Superconducting Circuits

by

Ruben van Staden

*Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University*

Supervisor: Prof. C. J. Fourie

March 2016

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .... March 2016

# Abstract


## Magnetic Field Visualization for Superconducting Circuits

R. van Staden

Thesis: MEng (EE)

March 2016

Ambient magnetic fields, trapped flux and fields generated by bias and ground return currents affect superconducting electronic circuits negatively. Analysis of current distribution in bias lines and in ground return paths gives us the ability to predict the behaviour of magnetic fields.

This work investigates the calculation of magnetic fields around superconducting integrated circuit structures from current density data, both for the interleaved uni-axial segment meshes of the FastHenry and FFH numerical solvers, as well as the more general tetrahedral and triangular meshes of more versatile electromagnetic solvers. A tool called MagnetEx is proposed, which gives the user a variety of visualization methods both for magnetic fields and current distribution. The result is a practical analysis tool with which the cause and effects of stray coupling or compromised shielding efficiency can be identified quickly and visually in superconducting circuit layouts.

# Acknowledgements

I would like to express my sincere gratitude to the following people:

- To my mother for her patience with me to make billions.
- Prof C. Fourie, my supervisor and Kyle Jackmann for inspiration and encouragement. You are the two most intelligent people I know.
- The Japanese professors and students at Yokohama University, specially Narama-san who helped me design the eSFQ circuits.

# Contents

# List of Figures

# Chapter 1

# Introduction

In this thesis a set of tools where developed for visualization of current distributions and magnetic fields in and around superconducting electronic circuits. Test results for magnetic fields and current distributions are shown for eSFQ cells that were designed and fabricated. It is still unclear to what effects ground plane holes and trapped flux have on circuit operations, and visualizing the magnetic fields and current distributions of a circuit is the first step to solving this problem. We discuss the different methods and algorithms used to calculate the magnetic fields around superconducting circuits. Results are given that shows how magnetic fields change due to hole placements with and without trapped flux, and also how these changes in the magnetic field results in changes in the current distribution.

Magnetic fields are calculated using both FFH [2], [3] and TetraHenry [4] in conjunction with the Biot-Savart's law. FastHenry and TetraHenry are MoM solvers for magnetoquasistatic problems using rectangular and terahedral meshing techniques, respectively. Using the FastHenry engine, InductEx [5] meshes the superconducting layers into three dimensional rectangles. Each layer is meshed to represent current in the $x$- and $y$-directions, as well as $z$-directions for vias. FastHenry has been modified to write a filament (.fil) file that contains all the information of each filament, including the current vector through each filament and the dimensions of the representing rectangle.

The data structure can be viewed as the 'road map' of any software package, which often dictates the organization and flow of the program, and thus requires very careful deliberation in the initial design of the package. Therefore, it is important to make the program as dynamic to future changes as possible, since we are not really sure what exactly to expect from field visualization and how this will be implemented into solving circuit problems more accurately. The reported toolchain is highly configurable with different algorithms implemented that can easily be iterated over by the designer.

# Chapter 2

# Geometric Processing

Any three-dimensional object can be represented as a set of closed polygon surfaces. In superconducting circuits we precisely define the volume features using rectangular or tetrahedral modeling methods. InductEx creates three-dimensional models [6], [7], and does the special interleaved rectangular meshes by FFH. For tetrahedral meshes, InductEx sends solid geometry models to third party tools such as GMSH [8] to generate the meshes.

Once each layer volume has been meshed, the input data (segments) are organized into data structures that are to be used in the processing and displaying of magnetic fields. The data structures contain the geometric properties of each individual segment, which contains boundary coordinates and parameters to identify the spatial orientation of the polygon volumes. All calculations are done on the true dimensions in and around the circuit layers. Then the results are scaled to fit the image coordinates. In the case of the magnetic fields, the spatial grid points at which the field values are calculated are mapped to the image coordinates. The geometric processing algorithms discussed and implemented into MagnetEx are explained in more details in [9].

## 2.1 Segmentation

In the reported tool a set of vertices that make up the smallest meshed polygon of a layer is defined as a segment. For segment operations the program must include capabilities for creating and deleting them. When a segment is created the coordinate positions and attribute values specified in the segment are stored as a structured variable. Different attributes can be assigned to a created segment. We will discuss the different methods used to handle rectangular and tetrahedral segments.

### 2.1.1 Line-Plane Intersection

Determine if a line with endpoints $q$ and $r$ intersects a plane. Just as all points on a line must satisfy a linear equation in $x$ and $y$, so too must all the points on a plane satisfy the plane equation

$$Ax + By + Cz = D \tag{2.1.1}$$

This can be described in vector form as

$$(x, y, z) \cdot (A, B, C) = D \tag{2.1.2}$$

The geometric meaning of this equation is that every point $(x, y, z)$ on the plane projects to the same length onto $(A, B, C)$. Any point $p(t)$ on the line can be represented by moving out to the $q$ endpoint, and then adding a scaled version of a vector along the line:

$$p(t) = q + t(r - q) \tag{2.1.3}$$

then using equation 2.1.2 we get the scaling factor $t$ as

$$\begin{aligned} p(t) \cdot (A, B, C) &= D \\ q \cdot N + t(r - q) \cdot N &= D \\ t(r \cdot N) &= D \\ t &= \frac{D - q \cdot N}{(r - q) \cdot N} \end{aligned}$$

### 2.1.2 Point Inside Segment

The spatial orientation of a segment polygon are obtained from the vertices and the equations that make up the polygon surface planes. The equation for a plane surface is

$$Ax + By + Cz + D = 0 \tag{2.1.4}$$

where $(x, y, z)$ is any point on the plane, as discussed above. The coefficients $A, B, C, D$ are constants that can be calculated using the coordinate values of three noncollinear points in the plane, and can be found using Cramer's rule

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix}$$

$$B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

$$D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

when expanding the coefficients, we get the following

$$\begin{aligned} A &= y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2) \\ B &= z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2) \\ C &= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \\ D &= -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1) \end{aligned}$$

Orientation of a plane surface in space is specified by the normal vector to the plane, which in this case has coordinates $(A, B, C)$. The side of the plane that faces the tetrahedron interior is called the "inside". If vertices are specified in a counterclockwise direction when viewing the outer side of the plane (looking into the tetrahedron center) in a right-handed coordinate system, the direction of the normal will be from the inside to the outside. In other words, the normal vector will point from the center of the tetrahedron to the outside. Using the plane equation, we can determine whether a point $(x, y, z)$ is inside or outside a plane. If the following inequality is satisfied, then the point lies outside the plane:

$$Ax + By + Cz + D > 0 \tag{2.1.5}$$

Using this relation we can implement a method to determine if the point is inside the tetrahedron as follow:

- Get the vector from the tetrahedral center to the face center, called $V_{ft}$.
- Get the normal of the face using the right hand rule, called $V_{fn}$.
- Get the dot product between $V_{ft}$ and $V_{fn}$.

If the result is larger than zero, then the face normal is pointing to the outside of the tetrahedron, if is smaller than zero, then the normal is pointing to the inside. In the reported tool we defined all surface normal vectors to point inwards. If they point outwards the order of the vertices are reversed. Once we have the normal of all faces pointing inwards, we can use the plane equation to determine on which side of the face the observation point is. If we define the normal of each face to point inwards, then if the plane equation is larger than zero for all four faces, we know the point lies inside the tetrahedron.

## 2.2 Polygon Dimensions

Scaling and translating is divided into two parts. One is calculating the scaling and translating factors of the circuit as seen from the cross-sectional side view. The second is the scaling and translating as seen from the top view.

### 2.2.1 Scaling Factor

Scaling is done by dividing the image width with that of the actual true circuit width. We find the scaling factor as a fraction of the real circuit dimensions with the image dimensions. The three-dimensional transformation matrix are expressed in the following form

$$\begin{bmatrix} D_x & 0 & 0 & 0 \\ 0 & D_y & 0 & 0 \\ 0 & 0 & D_z & 0 \\ K_x & K_y & K_z & 1 \end{bmatrix}$$

Factors $D_x, D_y, D_z$ are the ratios of the dimensions of the viewport and regular circuit dimensions in the $x$, $y$ and $z$ directions:

$$D_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$
$$D_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$
$$D_z = \frac{zv_{max} - zv_{min}}{d_f - d_n}$$

where the view volume boundaries are established by the window limits $(xw_{min}, xw_{max}, yw_{min}, yw_{max})$. The additive factors in the transformations are

$$K_x = xv_{min} - xw_{min} \cdot D_x$$
$$K_y = yv_{min} - yw_{min} \cdot D_y$$
$$K_z = zv_{min} - d_n \cdot D_z$$

### 2.2.2 Scaling

The matrix operation for scaling in three dimensions relative to the coordinate orientation is

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where scaling parameters $S_x, S_y, S_z$ are assigned any positive values. This matrix operation scales a point at $(x, y, z)$ to position $(x', y', z')$ with the scaling equations

$$x' = x \cdot S_x$$
$$y' = y \cdot S_y$$
$$z' = z \cdot S_z$$

When the above transformations is applied to defining points in an object, the object is scaled and moved relative to the coordinate origin. If the transformation parameters are not all equal, relative dimensions in the object are also changed. Therefore, we have to translate the scaled polygons to fit the image dimensions.

### 2.2.3 Translation

To scale an object with respect to a fixed point $(x_F, y_F, z_F)$ the following transformation matrix are used

$$T(-x_F, -y_F, z_F) \cdot S(S_x, S_y, S_z) \cdot T(x_f, y_F, z_F) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ (1 - S_x)x_F & (1 - S_y)y_F & (1 - S_z)z_F & 1 \end{bmatrix}$$

where $T$ represents the translation matrix and $S$ the scaling matrix.

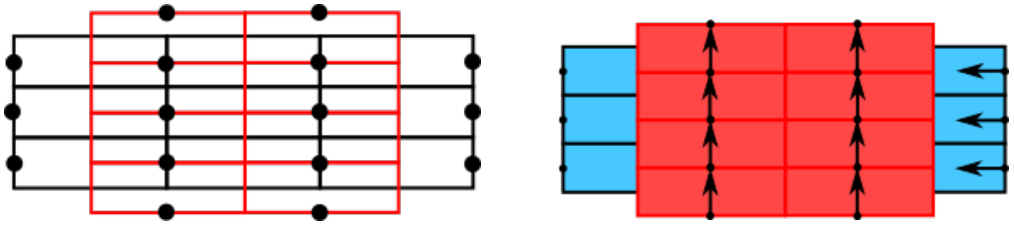
# Chapter 3

# Meshing

InductEx [10] is a wrapper for FFH and TH, providing pre- and postprocessing capabilities. Different numerical and analytical methods are used for calculating magnetic fields with different segment shapes. Each segment has its own current distribution vectors. Therefore, it is important to know the different attributes to assign to each segment data structure.

## 3.1 Rectangular Meshes for FFH

Polygon shapes are described by vertices and edges. Vertices are considered to be non-collinear points, that when connected by straight lines called edges, they uniquely define a polygon.



(a) $x$- and $y$-directed segments (b) Segments with current vectors

Figure 3.1: FFH Segmentation

InductEx meshes the circuit layers into rectangular segments, in the $x$-, $y$- and $z$-directions, when using the FFH back engine. The $x$- and $y$-directed segments account for non-uniform current flow across the structure, whereas $z$-directed segments model current flow through via structures. Each structure is meshed in both the $x$- and $y$-directions, as seen in Fig. 3.1a. Segments that are $x$-directed only has a $x$-directed current vector, and the same goes for the $y$-directed segments. The segments are placed so that their nodes connect (lay on top of each other), as shown in Fig. 3.1a. In Fig. 3.1b the current vectors

are added to the segments, where the blue segments are $x$-directed and the red segments are $y$-directed.

## 3.2 Tetrahedral Meshes for TH

InductEx converts geometric models into solid geometry representations that are meshed with tetrahedra by third-party tools such as GMSH and HyperMesh. This makes it possible to mesh tight angles inside the circuit with more accuracy. Current distribution over tetrahedra is not uniform, which makes it much more difficult to solve Biot-Savart, then rectangular segments. Here, we describe the different programming functions used to manipulate tetrahedrons to obtain the needed information for magnetic field calculations.

### 3.2.1 Tetrahedral Volume

A tetrahedron is described with the following four vertices:

$$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4 \tag{3.2.1}$$

where each vector $\mathbf{v}_i$ has a $x_i$, $y_i$ and $z_i$ component. The element has six edges and four faces. Edges are straight because they are defined by two corner points. Faces are planar because they are defined by three vertex points. At each corner three sides and three faces meet. The volume is given by the following determinant in terms of the corner coordinate values:

$$V = \frac{1}{6} \det \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \tag{3.2.2}$$

To insure that the volume is always positive the vertices must be non-coplanar and appropriately numbered. For correct numbering, first pick any corner as the initial one. Then pick a face to contain the first three corners. The opposite corner will be numbered 4. Finally, number those three corners in a counterclockwise direction when looking at the face from the outside, as explained in Chapter 2.

### 3.2.2 Face Center and Tetrahedral Center

The center points are required to determine the face orientation so that we can use the methods in Chapter 2 to determine if an observable grid point is inside or outside a segment. We ultimately want to distinguish between magnetic fields inside or outside a superconducting layer. Let us consider a tetrahedral

cell $\Omega = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ where the faces of $\Omega$ are four triangles $T_n(n = 1 - 4)$

$$T_1 = \{\mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$$
$$T_2 = \{\mathbf{v}_3, \mathbf{v}_1, \mathbf{v}_4\}$$
$$T_3 = \{\mathbf{v}_4, \mathbf{v}_1, \mathbf{v}_2\}$$
$$T_4 = \{\mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_2\}$$

Calculate the center of the tetrahedron using the follow equation

$$\mathbf{C}_{tet} = \frac{\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3 + \mathbf{v}_4}{4} \tag{3.2.3}$$

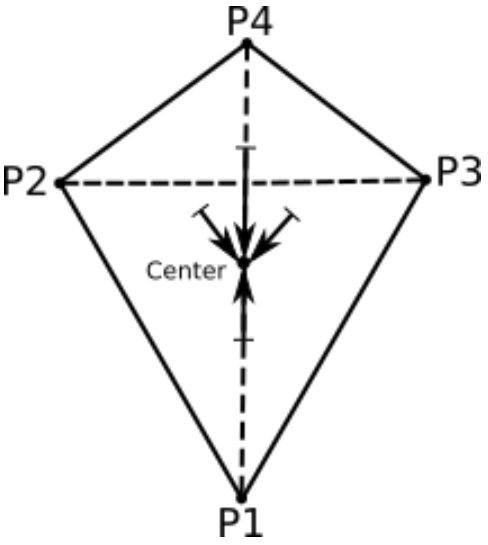
Calculate the center of face one, as follow

$$\mathbf{C}_{face} = \frac{T_1}{3} = \frac{\mathbf{v}_2 + \mathbf{v}_3 + \mathbf{v}_4}{3} \tag{3.2.4}$$

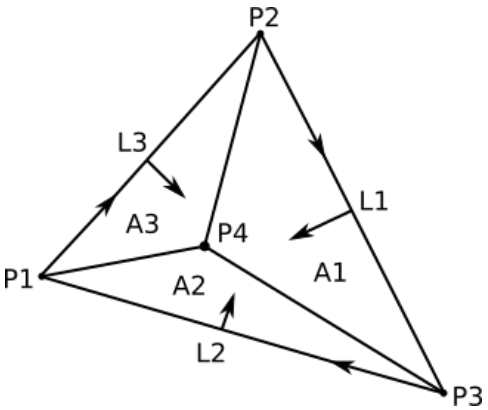
this algorithm can be used on all four faces.

### 3.2.3 Face Orientation

Another method that was implemented to determine if a point is inside a tetrahedron and arranging the face orientation is described in this section. We first see if the point's $y$-coordinate are within the edge's scope, then we check whether the point is to the left of any of the tetrahedra edges. If that is true, the line drawn rightwards from the test point crosses that edge.




(a) Tet Center

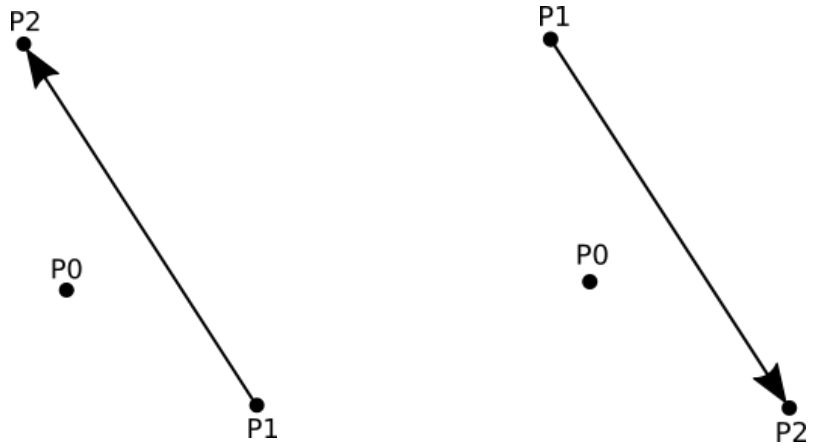



(b) Face Orientation

Figure 3.2: Tetrahedral Definitions

If the point lies left of all four faces of the tetrahedron, then the point is defined as inside the tetrahedron. In Fig. 3.3a the arrow represents the

vector between any two points of the two-dimensional tetrahedral face. A test is carried out to see if the third point is to the left of the represented vector, assuming a counterclockwise rotation.




(a) Counterclockwise (b) Clockwise

Figure 3.3: Triangle Orientation

To save the vertices of every face in a counterclockwise direction we define a vector between any two of the three points defining the face. An is-left test is then done, to see if the third point is on the left side of the defined vector. $P_0$ is always connected to $P_1$ and, $P_1$ to $P_2$. We want the $P_0$ node to always be on the left side of the segment line for each face. The thee faces connected to $P_0$ will automatically set the direction of the vertices for the fourth face. In Fig. 3.3a this will be a counterclockwise rotation with the point on the right side. In Fig. 3.3b it will be a clockwise rotation with the point on the left side.

# Chapter 4

# Current Distribution

Both FFH and TetraHenry calculates the current distribution density through each segment by assuming sinusoidal steady-state, and then applying the magnetoquasistatic assumptions that the displacement current, $\epsilon\omega\mathbf{E}$, is negligible everywhere [3]. Using Maxwell's equations we get

$$\nabla \times \mathbf{E} = -j\omega\mu\mathbf{H} \tag{4.0.1}$$
$$\nabla \times \mathbf{H} = j\omega\epsilon\mathbf{E} + \mathbf{J} \tag{4.0.2}$$
$$\nabla \cdot (\epsilon\mathbf{E}) = \rho \tag{4.0.3}$$
$$\nabla \cdot (\mu\mathbf{H}) = 0 \tag{4.0.4}$$

where $\omega$ is the angular frequency. Assuming that we are using an ideal current source, then the divergence of equation 4.0.2 gives

$$\nabla \cdot \mathbf{J} = I_s(\mathbf{r}) \tag{4.0.5}$$

Following the derivation in [3] we can show that the magnetic vector potential can be calculated using the current distribution density calculated by FFH or TetraHenry.

$$\mathbf{A}(\mathbf{r}) = \frac{\mu_0}{4\pi}\int \frac{\mathbf{J}(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}dV' \tag{4.0.6}$$

Using equation 4.0.6 and Biot-Savart's law we can calculate the magnetic field vectors at any point in space, as will be explained in the next chapter.

InductEx loops through each port in a circuit netlist, then excites one port while shorting all other ports to ground. The circuit inductances are calculated for each of these scenarios and the superposition of all these excited ports are taken. The same methods can be applied to see the current distribution through the circuit with any one port excited and the others shorted to ground. Superpositions between different excited ports can then be taken to get the resultant current spread. This can be of practical use to for instance see the instantaneous effect that an incoming SFQ pulse will have on the biasing current.

The main idea is to create a visualization tool that is dynamic and adaptable to any necessary changes. Then from these results we try to understand the concept of flux trapping. Trapped fluxons mutually connects to nearest superconducting layers, which then changes the magnetic fields around the layer and in turn change the critical current through the layer.

## 4.1 FFH

A drastically re-engineered program based on FastHenry, FFH, prints all necessary information concerning each segment, including the current density vector to a segmentation file. The $x$, $y$ and $z$ current density in each segment of the superconducting layer, is read in by the segmentation file. This file is then processed by the reported tool and a structure variable is created for each segment. If the current density inside each segment is assumed to be constant, then the approximation to the unknown current distribution can be written as

$$\mathbf{J}(\mathbf{r}) = \sum_{i=1}^{b} I_i \omega_i(\mathbf{r}) \mathbf{I}_i \tag{4.1.1}$$

where $I_i$ is the current inside segment $i$, $\mathbf{I}_i$ is a unit vector along the length of the segment and $\omega_i(\mathbf{r})$ is the weighting function which has a value of zero outside segment $i$, and $1/a_i$ inside, where $a_i$ is the cross-sectional area.

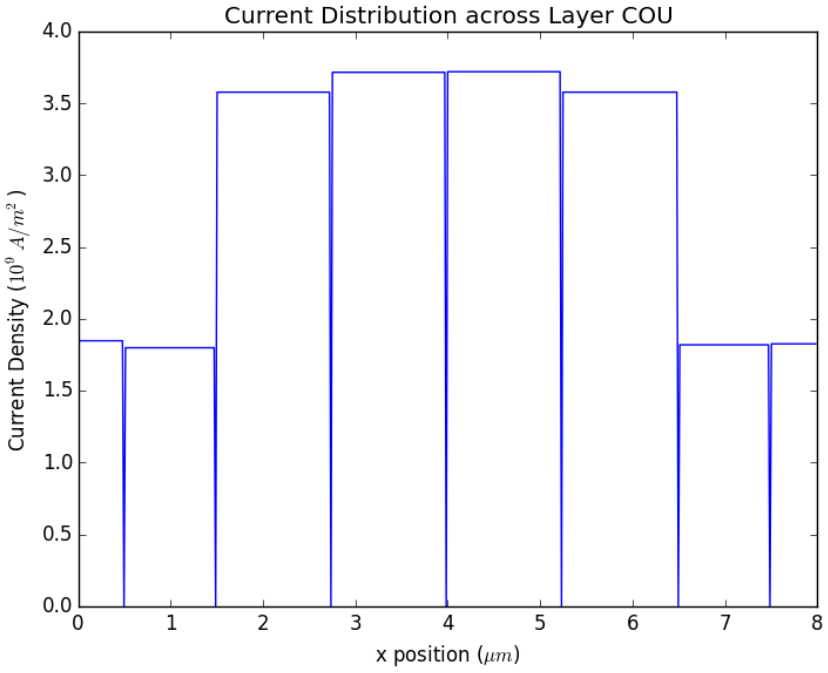
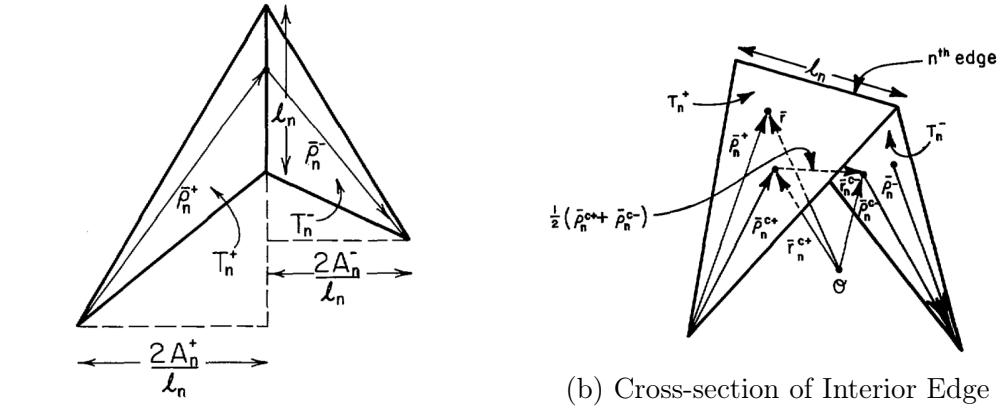



Figure 4.1: Current in GP segments from cross-sectional view

In some cases a layer is divided into more than one level of segments along the $z$-direction. An octree searching algorithm was implemented to find the segments that lie on top of each other. The resultant current density between these segments are found and recalculated using the new cross-sectional area of the combined segments. Quadtree and octree algorithms that was implemented are explained in Chapter 8. In Fig. 4.1 we see the cross-sectional current of a basic strip line, where the width of the histogram columns represents the current though that segment.

## 4.2 TetraHenry

Within each tetrahedron the current density is the sum of four linear vector functions, which is associated with each face. These functions can be combined to represent a constant vector in any direction. To determine which tetrahedrons are sliced, we loop through all four points of each tetrahedron and see if there is a vertex on both sides of the cross-sectional line. MagnetEx stores each segment in a structure that contains the location and dimensions of each segment, including the current vector and segment direction. This makes it easier to draw the cross-section view and debug magnetic fields.




(a) Interior Edge of Two Tetrahedrons

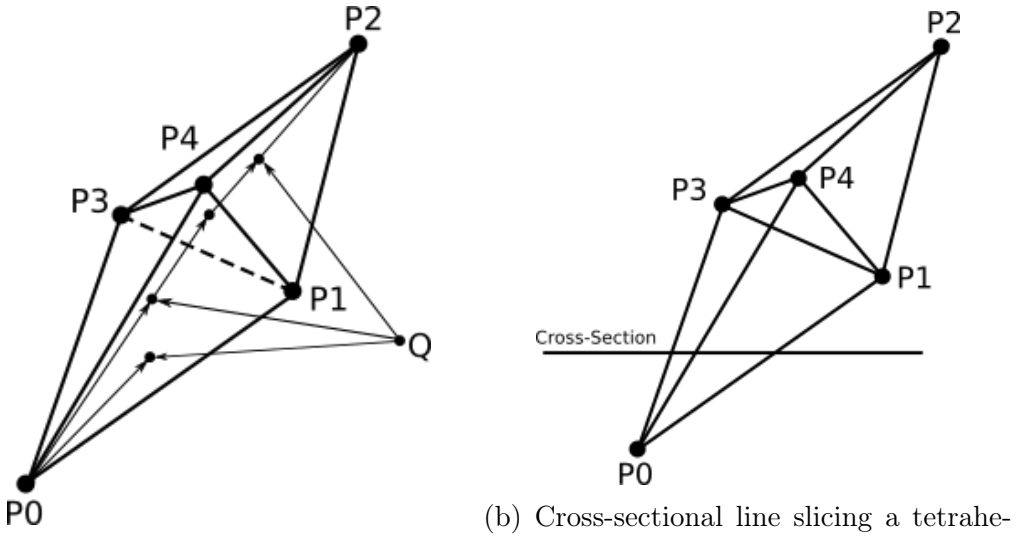
(b) Cross-section of Interior Edge

Figure 4.2: Basis Current Vectors of Tetrahedrons [1]

The surface charges will accumulate at the interfaces, due to the current not being continues across material interfaces of inhomogeneous dielectric bodies [11]. The charge density is constant within each tetrahedron and the currents are divergent free, which means the current flows linearly through each tetrahedron face.

Fig. 4.2 shows two triangles, $T_n^+$ and $T_n^-$, associated with the interior $n$th edge that connects them. In the three-dimensional case the two tetrahedrons

will share a face, Fig. 4.3. We explain the two-dimensional method for ease of understanding, since the equations in three-dimensions only differ by a constant and replacing the triangle area with the tetrahedral volume as explained in [12]. Points in $T_n^+$ may be designated either by the position vector $\rho_n^+$ defined with respect to the free vertex of $T_n^+$. Similar remarks apply to points in $T_n^-$ and the position vector $\rho_n^-$, except they are in opposite directions.




(a) Current vectors inside a tetrahedron

(b) Cross-sectional line slicing a tetrahedron

Figure 4.3: 3D Tetrahedron Segment

Taking the current direction into account is very important in correctly solving Biot-Savart's law, and is defined as being positive from $T_n^+$ to $T_n^-$. The current directions are saved inside the data structure that describes each individual segment. The current basis functions at the $n$th edge is

$$\mathbf{f}_n(\mathbf{r}) = \begin{cases} \frac{l_n}{2A_n^+}\rho_n^+ & \mathbf{r}\epsilon T_n^+ \\ \frac{l_n}{2A_n^-}\rho_n^- & \mathbf{r}\epsilon T_n^- \\ 0 & \text{otherwise} \end{cases} \tag{4.2.1}$$

where $l_n$ is the length of the edge and $A_n^\pm$ is the area of the triangle $T_n^\pm$. The current has no component normal to the boundary of the surface formed by the triangle pair $T_n^\pm$, and therefore no line charge exist along this boundary. The component of current normal to the $n$th edge is divergent free, which means it is constant and continuous across the edge. Therefore, the current flows continuously between segments without generating line charges. The current charge density is constant in each triangle and are given by the following

expressions

$$\nabla_s \cdot \mathbf{f}_n(\mathbf{r}) = \begin{cases} \frac{l_n}{A_n^+} & \mathbf{r}\epsilon T_n^+ \\ \frac{l_n}{A_n^-} & \mathbf{r}\epsilon T_n^- \\ 0 & \text{otherwise} \end{cases} \tag{4.2.2}$$

The current on the surface are given by

$$\mathbf{J} = \sum_{n=1}^{N} I_n f_n(\mathbf{r}) \tag{4.2.3}$$

where $N$ is the number of interior edges.

### 4.2.1 Current Density at Gaussian Quadrature Point

Each tetrahedron has four current vectors, one for each face. The current vectors starts at one of the four vertices and moves to the corresponding opposite face of the tetrahedral. The following equation is used to calculate the current density for each corresponding vertex $\mathbf{v}_i$ at a specific Gaussian quadrature point.

$$J_i = I_{imag} \cdot \mathbf{r} \tag{4.2.4}$$

Here $i = 1, 2, 3$ and 4 are the corresponding tetrahedral vertices, and $\mathbf{r}$ is the vector from the tetrahedral vertex to the quadrature point. The current density at this point is calculated and summed for all vertices. The result being the total or actual current density at that point inside the tetrahedron. This is done for four points inside each tetrahedron, which are then used in Biot-Savart's law to calculated the magnetic field at a specific observation point as explained in Chapter 5.

# Chapter 5

# Biot Savart

The Biot-Savart's law is a well known equation used to calculate the magnetic field vector at any point of the space for a given current distribution. The magnetic fields are calculated at specific points around the circuit layer dimensions, as seen from a cross-sectional view. These predefined points are selected by creating an evenly spaced two-dimensional grid around the circuit's cross-sectional view, Fig. 5.1.

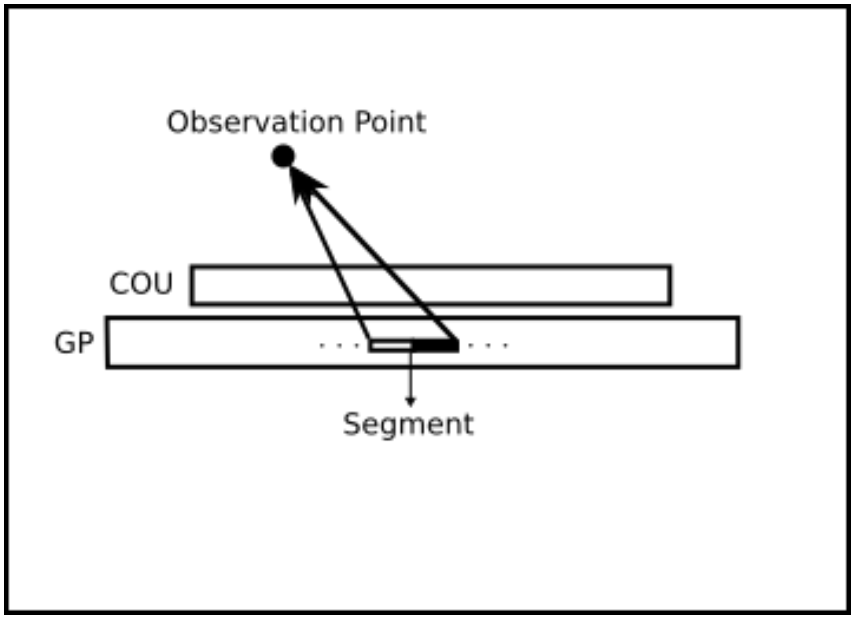



Figure 5.1: Magnetic Field Observation Point

These grid points are defined as the magnetic field's observation points. After the magnetic fields at each observable point is calculated, the observable points are mapped to each pixel on the image and each pixel are given a **rgb** color value corresponding to the magnetic field magnitude at that specific point. The observation grid is created by subdividing the entire space into

evenly sub-spaced points. The Clippers library is a polygon manipulation library used to perform line and polygon clipping operations. This library is used to unify all sliced segments, which will return the layer polygons as seen from the cross-sectional view. The library is then used again to find the difference between the grid points and the layer polygons. This will return only the grid points outside the layer polygons.

## 5.1 Solving Biot-Savart for FFH Current Distribution

The numerical methods for solving the magnetic field equations are not accurate enough for practical purposes. Therefore, the preferred solution is either using a Gaussian quadrature numerical solution or an analytical method [13]. Working with rectangular segments an analytical algorithm was implemented, since it has been well studied and documented. The magnetic fields are calculated using the law of Biot-Savart, which in terms of the magnetic vector potential can be written as

$$\mathbf{B} = \nabla \times \mathbf{A} \tag{5.1.1}$$

where the magnetic vector potential is given by

$$\mathbf{A}(\mathbf{r}) = \frac{\mu_0}{4\pi} \int \frac{\mathbf{J}(\mathbf{r}')}{R} dV' \tag{5.1.2}$$

Here $\mathbf{J}$ is the current vector at the point $\mathbf{r}'$ in the superconducting layer and $R = |\mathbf{r} - \mathbf{r}'|$ is the distance between the current point and the observation point. Biot-Savart can also be solved using the magnetic field $\mathbf{H}$ in terms of Green's function [14]

$$\mathbf{H}(\mathbf{r}) = \frac{1}{4\pi} \int_V \mathbf{J}(\mathbf{r}') \times \nabla' G(\mathbf{r}, \mathbf{r}') dV' \tag{5.1.3}$$

where $\nabla' G$ is the gradient of Green's function in free space,

$$\nabla' G(\mathbf{r}, \mathbf{r}') = \frac{\mathbf{r} - \mathbf{r}'}{|\mathbf{r} - \mathbf{r}'|^3} \tag{5.1.4}$$

### 5.1.1 Analytical Solution

An analytical method [15], [13] to calculate Biot-Savart's law was implemented for calculating the magnetic fields using the FFH engine. The magnetic field at point $\mathbf{r}$, is the sum of the magnetic field caused by every segment in the list of specified layers. Let $\mathbf{s} = \mathbf{r} - \mathbf{r}$' then

$$\mathbf{B}(\mathbf{r}) = \frac{\mu_0}{4\pi} \sum_{n=1}^{N_{seg}} \int_{V_n} \frac{\mathbf{J}_n \times \mathbf{s}}{s^3} dV \tag{5.1.5}$$

where, $n$ is the specific segment, $\mathbf{J}_n$ is the current density through that segment, and $V_n$ is the segment volume. Because of uniform current density, equation 5.1.5 becomes

$$\begin{aligned}\int_{V_n} \frac{\mathbf{J}_n \times \mathbf{s}}{s^3} dV &= \mathbf{J}_n \times \int_{V_n} \frac{\mathbf{s}}{s^3} dV \\ &= \Theta(s)|_{V_n}\end{aligned}$$

which is analytically solved as

$$\Theta_i(s) = s_j - s_i \arctan(\frac{s_j}{s_i}) + s_i \arctan(\frac{s_j s_k}{s_i \epsilon}) - s_k \log(2(s_j + \epsilon)) - s_j \log(2(s_k + \epsilon)) \tag{5.1.6}$$

where $\epsilon = (s_i^2 + s_j^2 + s_k^2)^{1/2}$. The letters $i$, $j$ and $k$ are cyclic permutations of the $x$, $y$, and $z$ coordinates of the eight corners of each segment.

## 5.2 Solving Biot-Savart for TetraHenry Current Distribution

Volume integration over the tetrahedron is done using a Gaussian quadrature algorithm. Quadrature rules [16] are developed for exactly integrating products of polynomials and generalizes functions over tetrahedral domains. Applying the quadrature algorithm we loop through each observable point in space and see the effect each tetrahedron has on this specific point. We implemented a four point quadrature rule, which means there are four points for each tetrahedron that will be used to calculate the magnetic field at an observation point. Exact quadrature rules are derived for triangular elements that have a linearly varying current distribution and quadratic integrands.

### 5.2.1 Numerical Solution

To determine the magnetic field induced at a specific point by the current density inside a tetrahedron, we have to apply the Gaussian quadrature algorithm [16]. The algorithm chooses four points in each tetrahedron using a weighting algorithm, which is directly dependent on the current vectors through the tetrahedron.

Biot-Savart's law is applied to all four points inside the tetrahedron to determine the magnetic field at each observable point in space. According to Biot-Savart's law the magnetic field at point $\mathbf{r}$ is given by

$$\mathbf{B} = \frac{\mu_0}{4\pi} \int_{V'} \frac{\mathbf{J}(\mathbf{r}') \times (\mathbf{r} - \mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|^3} \tag{5.2.1}$$

Singularities in the integral is avoided by not taking observation points within the superconducting layers. Therefore, the Gaussian quadrature becomes

$$\int_{-1}^{1} f(r)dr = \sum_{i=1}^{n} \omega_i f(r_i) \tag{5.2.2}$$

Equation 5.2.2 will correspond to the integral in the Biot-Savart equation given by 5.1.5. The default value of $n$ is 4, and is called the order of quadrature and the corresponding quadrature weights are

$$\omega_i = \{\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\}^t \tag{5.2.3}$$

To understand how this method was implemented in code, we have to understand how it was derived and how it works in a discrete spatial domain. Assume the function $f$ is a two dimensional line, then let the one-dimensional integral be calculated as

$$I = \int_{-1}^{1} f(r)dr \tag{5.2.4}$$

The simplest way to evaluate $I$ is to sample $f$ at the middle point and multiply by the length of the interval, to obtain

$$I = 2f_1 \tag{5.2.5}$$

If the curve was a straight line, then the result would be correct. The discrete form of the equation is given by

$$\begin{aligned} I &= \int_{-1}^{1} f(r)dr \\ &= \sum_{i=1}^{n} \omega_i f(r_i) \end{aligned}$$

where $\omega_i$ is called the weight associated with the $i$th point, and $n$ is the number of sampling points. To evaluate the integral, we evaluate the function at several sampling points, multiply each value $f_i$ by an appropriate weight $\omega_i$, and add the results together. In the Gauss method the location of sampling points is such that for a given number of points the greatest accuracy is obtained. The sampling points are located symmetrically about the center of the interval. The weight would be the same for symmetrically located points. Thus, for example, if we use the three-point Gaussian formula, we get

$$I = 0.555556f_1 + 0.888889f_2 + 0.555556f_3 \tag{5.2.6}$$

this is the exact result if $f(r)$ is a polynomial of order less than or equal to five, which in our case will be a four point tetrahedron. The principle involved

in deriving the Gauss quadrature formula can be illustrated by considering a simple function

$$f(r) = a_1 + a_2 r + a_3 r^2 + a_4 r^3 \tag{5.2.7}$$

where $a_i$ are the polynomial constants. If $f(r)$ is integrated between $-1$ and 1, the area under the curve is

$$I = 2a_1 + \frac{2}{3}a_3 \tag{5.2.8}$$

By using two symmetrically located points $r = \pm r_i$, the area can be calculated as

$$\begin{aligned} I &= \omega \cdot f(-r_i) + \omega \cdot f(r_i) \\ &= 2\omega(a_1 + a_3 r_i^2) \end{aligned}$$

If we want to minimize the error $e = I - I$ for any value of $a_1$ and $a_3$, we must have

$$\frac{\partial e}{\partial a_1} = \frac{\partial e}{\partial a_3} = 0 \tag{5.2.9}$$

which will give

$$\omega = 1 \tag{5.2.10}$$

#### 5.2.1.1 Two Dimensional Quadrature

The integrals involved for triangular elements would be in terms of triangular or area coordinates and the following Gauss formula has been developed by Hammer and Stroud [17]

$$\begin{aligned} I &= \int\int_A f(L_1, L_2, L_3) dA \\ &= \sum_{i=1}^{n} \omega_i f(L_1^{(i)}, L_2^{(i)}, L_3^{(i)}) \end{aligned}$$

where we use a quadratic triangle, $n = 3$:

$$\begin{aligned} \omega_1 = \frac{1}{3}: &\quad L_1^{(1)} = L_2^{(1)} = \frac{1}{2}, L_3^{(1)} = 0 \\ \omega_2 = \frac{1}{3}: &\quad L_1^{(2)} = 0, L_2^{(2)} = L_3^{(2)} = \frac{1}{2} \\ \omega_3 = \frac{1}{3}: &\quad L_1^{(3)} = L_3^{(3)} = \frac{1}{2}, L_2^{(3)} = 0 \end{aligned}$$

where $L_1$, $L_2$ and $L_3$ are the normalized volume coordinates of the tetrahedron and must add to unity. The locations of the integration points are given in Fig. 5.2a.

#### 5.2.1.2 Three Dimensional Quadrature

For tetrahedral regions, four volume coordinates are involved and the integral can be evaluated as in the two-dimensional case:

$$I = \sum_{i=1}^{n} \omega_i f(L_1^{(i)}, L_2^{(i)}, L_3^{(i)}, L_4^{(i)}) \tag{5.2.11}$$

where we use a quadratic tetrahedron, $n = 4$:

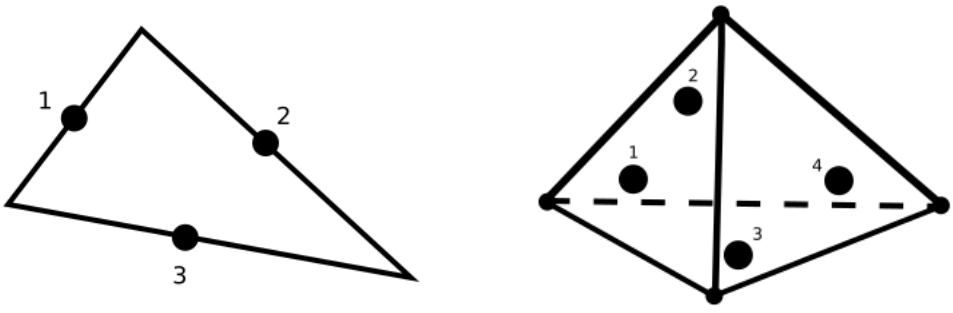
$$\begin{aligned}
\omega_1 = \frac{1}{3}: &\quad L_1^{(1)} = a, L_2^{(1)} = L_3^{(1)} = L_4^{(1)} = b \\
\omega_2 = \frac{1}{3}: &\quad L_2^{(2)} = a, L_1^{(2)} = L_3^{(2)} = L_4^{(2)} = b \\
\omega_3 = \frac{1}{3}: &\quad L_3^{(3)} = a, L_1^{(3)} = L_2^{(3)} = L_4^{(3)} = b \\
\omega_4 = \frac{1}{3}: &\quad L_4^{(4)} = a, L_1^{(4)} = L_2^{(4)} = L_3^{(4)} = b
\end{aligned}$$

with

$$a = 0.58541020 \tag{5.2.12}$$

and

$$b = 0.13819660 \tag{5.2.13}$$




(a) Triangle Quad Points

(b) Tetrahedron Quad Points

Figure 5.2: Qaudrature Points

All data are given by TetraHenry in a segmentation file that are read in by the program and stored inside a list of segment data structures. Special care has to be taken for the current direction when it enters or exits a tetrahedral face to go from or to another tetrahedron.

### 5.2.2 Analytical Solution

There are analytical methods for solving Biot-Savart over a two-dimensional triangle [18], [19], but for three-dimensional tetrahedrons no documented papers have been found. To solve the magnetic potential $\mathbf{A}$ integral over a tetrahedron we have to consider a distribution of sources within a region $V$. The boundary of $V$ is denoted by $\partial V$ and has an outward unit normal $\mathbf{n}$. $\partial V$ comprises a number of faces, the $j$th one of which is designated $\partial_j V$. The $j$th face, in turn, is a polygon having a boundary $\partial\partial_j V$ which comprises a number of edges, the $i$th of which is designated $\partial_i\partial_j V$. Gauss integral theorems are first used to transform integrals over $V$ into integrals over $\partial V = \sum_j \partial_j V$. Then integrals over $\partial_j V$ are transformed into integrals over the polyhedron edged, $\partial_i\partial_j V$. The current distribution through the tetrahedron must be linearly varying:

$$\begin{aligned}
\int_V \frac{1}{R}dV' &= \lim_{\delta\to 0}\frac{1}{2}\int_{V-V_\delta}\nabla'\cdot\mathbf{R}dV' + \lim_{\delta\to 0}\int_{V_\delta}\frac{1}{R}dV' \\
&= \lim_{\delta\to 0}\frac{1}{2}\int_{\partial(V-V_\delta)}\mathbf{R}\cdot\mathbf{n}dS' + \lim_{\delta\to 0}\frac{\delta^2}{2}\Omega(\mathbf{r}) \\
&= \frac{1}{2}\sum_j\int_{\partial_j V}\mathbf{R}\cdot\mathbf{n}dS' \\
&= -\frac{1}{2}\sum_j d_j\int_{\partial_j V}\frac{1}{R}dS' \\
&= \frac{1}{2}\sum_j d_j\Bigg\{\sum_i \mathbf{P}^0_{ij}\cdot\mathbf{u}_{ij}\times\Bigg[|d_j|\Bigg(\tan^{-1}\Bigg(\frac{P^0_{ij}l^+_{ij}}{(R^0_{ij})^2+|d_j|R^+_{ij}}\Bigg) \\
&\quad -\tan^{-1}\Bigg(\frac{P^0_{ij}l^-_{ij}}{(R^0_{ij})^2+|d_j|R^-_{ij}}\Bigg)\Bigg) - P^0_{ij}\ln\frac{R^+_{ij}+l^+_{ij}}{R^-_{ij}+l^-_{ij}}\Bigg]\Bigg\}
\end{aligned} \tag{5.2.14}$$

where $\mathbf{R}$ is the unit vector from $\mathbf{r}$ to $\mathbf{r}'$, and $S$ is the polygon in which the current distribution has a linear variation. A more detailed explanation of the equation is given in [18].

It is mathematically impossible to write a programming algorithm to solve this equation, therefore equation 5.2.14 will have to be re-derived for Biot-Savart's law. This is beyond the scope of this thesis and the numerical method is accurate enough. There are other derivations of equation 5.2.14 [20] which consist of more than 24 terms.

# Chapter 6

# Linear Integral Convolution

The Linear Integral Convolution (LIC) algorithm takes as input an image and a vector field defined on the same domain [21]. The output image is computed as a convolution of the intensity values over the integral curves of the vector field. In this case we want to visualize the topology of the magnetic field, the input image needs to have pixels uniformly distributed and with mutually independent intensities. White noise is used as the input image, which will then be convolved with the vector field. The LIC algorithm is a general method used to visual flow fields. Different enhancing and filtering techniques are used to improve the the visualization of the individual field lines.

## 6.1 DDA Convolution

The Digital Differential Analyzer (DDA) is an algorithm for calculating pixel positions along a line, using the straight line equation

$$y = mx + c \tag{6.1.1}$$

The LIC is a generalization of what is known as the DDA convolution. The DDA algorithm performs convolution on a line direction rather than on integral curves. For each pixel location $(i, j)$ on the input image $I$, we want to compute the pixel intensity in the output image $O(i, j)$. The DDA takes the normalized vector $V(i, j)$ corresponding to that location and moves in its positive and negative directions some fixed length $L$, which generates a line of locations

$$l(s) = (i, j) + sV(i, j), \qquad s\epsilon\{-L, L\} \tag{6.1.2}$$

and a line of pixel intensities $I(l(s))$ of length $2L+1$. The line function $I(l(s))$ is filtered and normalized to generate the intensity output $O(i, j)$:

$$O(i, j) = \frac{1}{2L+1} \sum_{s=-L}^{L} I(l(s))k(l^{-1}(i, j) - s) \tag{6.1.3}$$

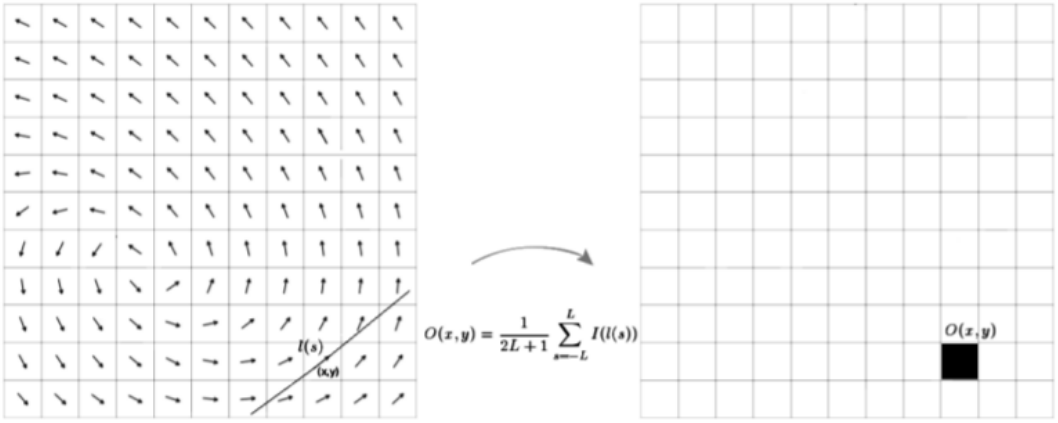



Figure 6.1: DDA Convolution

Here $k$ is a box filter and so the discrete convolution of the equation above becomes the average sum of pixels $I(l(s))$:

$$O(i,j) = \frac{1}{2L+1} \sum_{s=-L}^{L} I(l(s)) \tag{6.1.4}$$

## 6.2 Linear Integral Convolution

An integral curve of the vector $v$, passing over point $x_0$ at time $\tau_0$, is defined as a function $c_{x_0}$ over the domain $[-L, L]$ with:

$$\frac{d}{d\tau} c_{x_0}(\tau) = v(c_{x_0}(\tau)), c_{x_0}(0) = x_0 \tag{6.2.1}$$

From [21] the $\tau$ can be replaced by an arc length $s$, then the integral curve can be re-defined as follow

$$\frac{d}{ds} c_{x_0}(s) = \frac{v(c_{x_0}(s))}{|v(c_{x_0}(s))|}, c_{x_0}(s_0) = x_0 \tag{6.2.2}$$

The new output pixel $0(i,j)$, with $s_0$ such that $c_{(i,j)}(s_0) = (i,j)$, is computed by LIC as:

$$O(i,j) = \frac{1}{2L+1} \sum_{s=s_0-L}^{s_0+L} I(c_{(i,j)}(s)) k(s_0 - s) \tag{6.2.3}$$

and using a simple box filter

$$O(i,j) = \frac{1}{2L+1} \sum_{s=s_0-L}^{s_0+L} I(c_{(i,j)}(s)) \tag{6.2.4}$$

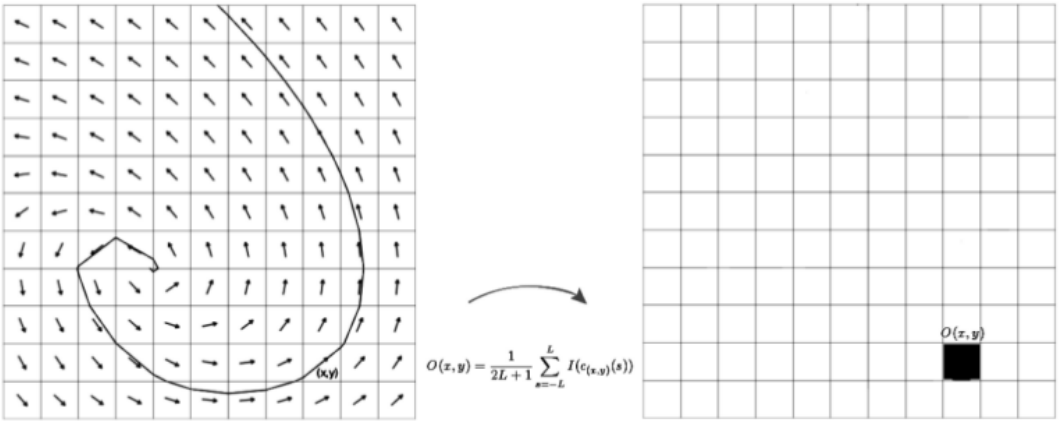



Figure 6.2: LIC Convolution

### 6.2.1 Normalization

Normalizing the convolution integral insures that the apparent brightness and contrast of the resultant image is well behaved as a function of kernel shape, phase and length. The magnetic field vector at pixel location of the normalization equation is divided by the integral of the convolution kernel. This insures that the normalized area under the convolution kernel is always unity resulting in a constant overall brightness for the image independent of the filter shape and LIC length. The normalization makes it easy to convert the result to an **rgb** color value to represent the magnetic field intensity in log scale.

## 6.3 Fast LIC Algorithm

A separate stream line segment and a separate convolution integral are computed for each pixel in the output image. A single stream line covers lots of image pixels and for a constant filter kernel $k$ very similar convolution integrals occur for pixels covered by the same stream line.

Given that when computed, an integral curve covers lot of pixels, uniqueness of the solution of the order differential equation implies that the convolution involved in LIC can be reused. Choose a box filter kernel and suppose we have an integral curve of a location $(i, j)$, say $c_{(i,j)}$ and another location along it $c_{(i,j)}(s)$, then their output values are related by

$$O(c_{(i,j)}(s)) = O(i,j) - \sum_{s'=s_0-L}^{s_0-L+s} I(c_{(i,j)}(s')) + \sum_{s'=s_0+L}^{s_0+L+s} I(c_{(i,j)}(s')) \qquad (6.3.1)$$

Fig. 6.3 and Fig. 6.4 illustrates this relation. To design a fast LIC algorithm [22], we have taken an approach which relies on computing the convolution integral by sampling the input texture $T$ at evenly spaced locations along

a pre-computed stream line. For each sample point the corresponding output image pixel is determined and the current intensity is added to that pixel. In this way we efficiently obtain intensities for many pixels covered by the same steam line. The probability for an output image pixel to be hit by a sample

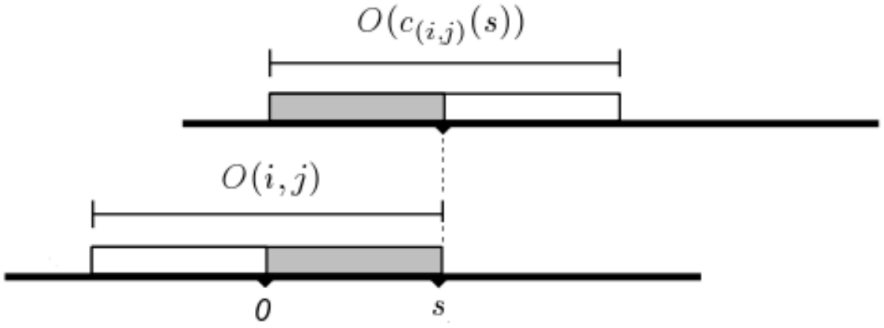



Figure 6.3: Reused Pixels in Grey

point is proportional to the length of the line segment covering that pixel. If a pixel has been visited a new stream line is computed, otherwise that pixel is skipped. In practice, to reuse an already computed convolution for a set of

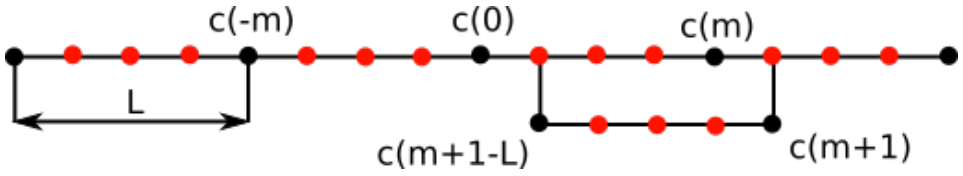



Figure 6.4: Program Implementation of Fig. 6.3

pixels, a matrix of the same size of the image is created such that each entry stores a one if that pixel has been visited, otherwise a zero is stored. The order in which the pixels are analyzed is important for the efficiency of the process. The goal is to hit as many uncovered pixels with each new integral curve to reuse the convolution as possible. For instance, we take the first pixel of each line and make the calculations, then the second pixel and so on.

The discrete algorithm form of equation 6.3.1 are given below. After having computed $O(x_0)$, we step in both directions along the current stream line, thereby updating the convolution integrals as follows

$$O(x_{m+1}) = O(x_m) + k[I(x_{m+1+L}) - I(x_{m-L})]$$
$$O(x_{m-1}) = O(x_m) + k[I(x_{m-1-L}) - I(x_{m+L})]$$

Another thing to consider when implementing LIC is that the domain of the input $I$ and output image $O$ can be taken as continuous domains rather than

a grid of pixels $(i, j)$. Basically we take a continuous rectangular domain and define a set of cells with the center at pixel location $(i, j)$. Then to compute the output pixel at location $(i, j)$ one choose a number of sample locations within its correspondent cell, perform the computations and compute an average intensity value. Finally, when performing the convolution on pixels near the boundary of the image domain, sometimes the algorithm will try to retrieve an intensity value of an invalid pixel location, because the integral curve will leave the image domain. This problem can be solved by padding the image with zeros on the boundary.

## 6.4 Enhanced LIC Algorithm

There are many other improvements that can be made to the LIC method and visualizing magnetic fields in general, such as the Lagrangian-Eulerian approach, noise-based advection, coordinate integration, edge treatment, noise injection, coordinate reinitialization, noise blending, postprocessing and parallization [23], [24].

Parallelization is performed in image space rather than in time, where the image space coherence is exploited using a flexible update and communication scheme. Another more advanced implementation to improve the LIC algorithm speed is to use an advanced evenly-space streamline placement algorithm [25]. A fourth-order Runge-Kutta integrator with adaptive step size and error control can be employed for rapid accurate streamline advection. Cubic Hermite polynomial interpolation with large sample-spacing can also be adopted to create fewer evenly-space samples along each streamline to reduce the amount of distance checking.

### 6.4.1 Double LIC

The nature of the LIC algorithm is to average texel values that lie on an integrated path of a stream line in a vector field. When the input texel values of the LIC algorithm are random white noise, the histogram of the LIC image results in a distribution curve with a standard deviation range of approximately 100 to 150, for gray scale (256) levels. To improve the image quality we can alter the length of convolution in the LIC algorithm. When longer convolution length's are used, a greater number of texels on a particular vector line are assigned similar texel color values, but this increases the LIC computation time by a factor of four. Another method is to execute the LIC algorithm twice, where the output image of the first execution becomes the input image to the second execution. The second execution of the LIC algorithm creates the flow lines, not by averaging random white noise values, but by averaging color values that are related to a particular flow line, either light or dark.

### 6.4.2 Filtering

It is desirable to create periodic low-pass filters to blur the underlying texture in the direction of the vector field. A Hanning filter has this property. It has low band-pass filter characteristics, it is periodic by definition and has a simple analytic form. Since the LIC algorithm is by definition a local operation, any filter used must be windowed. Using a box window with unit height over a phase shifted Hanning filter the cutoff at the ends of the filter are noticeable. One solution to this problem is to use a Guassian window as suggested by Gabor [24]. By windowing the Hanning function by a Gaussian these cutoffs are smoothly attenuated to zero.

The histogram equalization filter moves texel values toward the extremes of the range. It allows for a wider resultant range of multiple levels. To increase the distinguishability between individual lines we add a $3 \times 3$ high-pass filter, where a local region rather than a pixel by pixel approach to image enhancement is needed. By altering texel values based on neighbouring values, we can increase the sharpness of each flow line.

### 6.4.3 Discrete Wavelet Transform for Post Processing

A novel image interpolation algorithm using the Discrete Wavelet Transform (DWT) [26] are used for post-processing to improve the clarity of the LIC field lines. This method preserves much of the sharp edge features in the image, and lessens the amount of color artifacts.

The DWT is a filter and re-sampling based method. As discussed in [27], re-sampling is the process of transforming discrete image pixels defined at one coordinate system to a new coordinate system of a different resolution. The re-sampling technique is used to up-sample an image to enhance its resolution. The re-sampling from one discrete signal $x[n]$ to another $y[n']$ of a different resolution is calculated as

$$y[n'] = \sum_{t=-w}^{w} h[t]x[n' - t] \tag{6.4.1}$$

where $h[t]$ is the interpolating function and $w$ is the desired filtering window. The above equation is a simple convolution operation of $x[n]$ with $h[t]$.

DWT can be implemented by filtering operations with well-defined filter coefficients [28]. In traditional convolution based approach to compute forwards DWT, the input signal $x$ is filtered separately by a low-pass filter $h$ and a high-pass filter $g$ [29]. The two output streams are then sub-sampled by simply dropping the alternate output samples in each stream to produce the

low-pass $y_L$ and high-pass $y_H$ subband outputs. Given a discrete signal $x(n)$, the output signals can be computed as follow:

$$y_L(n) = \sum_{i=0}^{\tau_L - 1} h(i) \times x(2n - i)$$

$$y_H(n) = \sum_{i=0}^{\tau_H - 1} g(i) \times x(2n - i)$$

where $\tau_L$ and $\tau_H$ are the lengths of the low-pass and high-pass filters, which in our case will be $3 \times 3$. For inverse transform, both signal outputs are first filtered by low-pass and high-pass filters and then added to obtain the signal $x'$.

Image signals are two-dimensional signals, and since most of the practical two-dimensional wavelet filters are separable functions, the 2D DWT can be implemented by applying the 1D DWT row-wise to produce an intermediate result and then applying the same 1D DWT column-wise on the intermediate result.

In the first step the image is decomposed into four subbands $LL$, $HL$, $LH$ and $HH$ by DWT, resulting in the wavelet coefficient image $I_{DWT}$. Here $LL$ mean applying a low-pass filter twice, and $LH$ means applying a low-pass then high-pass filter. The $HL$ and $LH$ subbands contain edge information in horizontal and vertical directions. The second step is to form a new wavelet coefficient image $I'_{DWT}$. We call it a virtual DWT image, whose $LL$ subband is nothing but the original input image $I$ with each pixel multiplied by a scaling factor. The dimensions of the new $LL$ subband is the same as the resolution $m \times n$ of the original image, $I$. The $HH$ subband of the virtual DWT image $I'_{DWT}$ is set all zeros with dimensions $m \times n$. The new $HL$ and $LH$ subbands of the virtual DWT image are generated from the original $HL$ and $LH$ subbands. In the last step we inverse transform (IDWT) this virtual DWT image to generate the desired enhanced image.

# Chapter 7

# Interpolation

Interpolation is the problem of approximating the value of a function for a non-given point in some space when given the value of that function in points around that point. Different interpolation algorithms can be implemented to solve magnetic fields calculation problems. In this chapter we will describe the different methods implemented and tested into the reported toolkit. One other method worth investigating is that of a multilevel Green's function interpolation method [30]. This method can increase the speed of solving Biot-Savart even faster than bicubic interpolation, by speeding up the matrix-vector multiplications in the iterative solution in which a computational complexity of $O(N)$ is achieved. The interpolation kernel $k(i)$ defines the list of neighbors to be considered and the weight assigned to them for calculating the value of the central pixel. Mathematically this corresponds to the operation:

$$g(x') = \sum_i f(x+i)k(i) \tag{7.0.1}$$

The simplest kernel is the nearest neighbor kernel, which corresponds to a box. This implementation by convolution has several advantages:

- It provides a uniform way to implement many different types of interpolation by choosing a suitable convolution kernel.
- It is easy to extend this method to different scaling and different dimensions.

## 7.1 Non-Adaptive Algorithms for Image Interpolation

In non-adaptive image interpolation algorithms, certain computations are performed indiscriminately to the whole image for interpolation regardless of its content. In other words, only certain magnetic field values around the defined area are taken into account.

### 7.1.1 Nearest Neighbor

The nearest neighbor algorithm selects the value of the nearest point and does not consider the values of neighboring points at all, yielding a piecewise-constant interpolant.

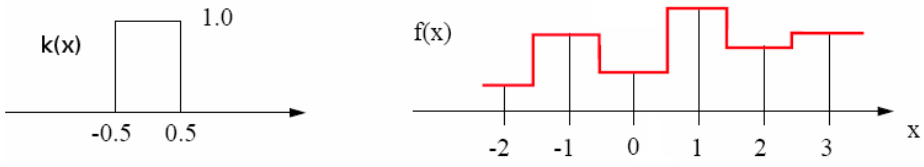



Figure 7.1: Nearest Neighbor Kernel

The convolution defines a general principle for interpolation. This is the simplest possible interpolation algorithm. Nearest neighbor selects the value of the pixel by rounding the coordinates of the desired interpolation point:

$$f(x) = \begin{cases} f(|x|) & \text{for } x - |x| < 1/2 \\ f(|x| + 1) & \text{otherwise} \end{cases}$$

As a result of this simplistic interpolation scheme, nearest neighbor doesn't have subpixel accuracy and generates strong discontinuities, especially when arbitrary rotations and scale changes are involved. The only interesting property of this algorithm is the fact that it preserves the original noise distribution in the transformed image.

### 7.1.2 Linear Interpolation

Linear interpolation uses a convolution kernel $k(x)$ which has the shape of a triangle. The output image thus has a smoother surface because the discretization is less strong.

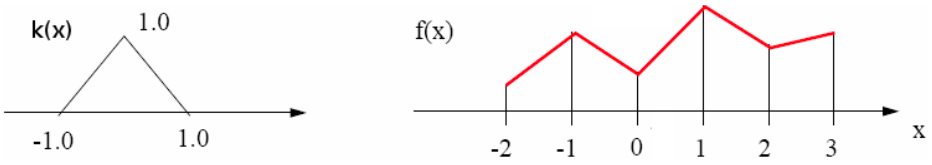



Figure 7.2: Linear Kernel

This is an interpolation method of curve fitting using linear polynomials. If the two known points are given by the coordinates $(x_0, y_0)$ and $(x_1, y_1)$, the

linear interpolant is the straight line between these points. For a value $x$ in the interval $(x_0, x_1)$, the value $y$ along the straight line is given from the equation

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \tag{7.1.1}$$

Solving this equation for $y$, with the known value at $x$, gives

$$y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0} \tag{7.1.2}$$

which is the formula for linear interpolation in the interval $(x_0, x_1)$. Linear interpolation is very accurate in one-dimensional problems. In our case we want to interpolate in two-dimensions, along the $x$- and $y$-directions.

### 7.1.3 Bilinear Interpolation

The bilinear algorithm interpolates from the nearest four mapped source pixels. The pixel mapping distance is set by the user, the smaller this distance, the more accurate the solution. It builds and evaluates two linear interpolation functions, one in the $x$-direction and one in the $y$-direction. If $f$ is a two-dimensional image, the bilinear interpolation equations for an arbitrary point $(x, y)$ are the following:

$$\begin{aligned} f_{y1} &= f_{11} + \frac{f_{21} - f_{11}}{x_2 - x_1}(x - x_1) \\ f_{y2} &= f_{12} + \frac{f_{22} - f_{12}}{x_2 - x_1}(x - x_1) \\ f(x, y) &= f_{y1} + \frac{f_{y2} - f_{y1}}{y_2 - y_1}(y - y_1) \end{aligned}$$

with:

$$\begin{aligned} x_1 &= |x|, & f_{11} &= f(x_1, y_1) \\ x_2 &= |x| + 1, & f_{12} &= f(x_1, y_2) \\ y_1 &= |y|, & f_{21} &= f(x_2, y_1) \\ y_2 &= |y| + 1, & f_{22} &= f(x_2, y_2) \end{aligned}$$

The main drawbacks of bilinear interpolation are poor preservation of image detail. For a unit square where we choose a coordinates system in which the four points are $(0, 0), (0, 1), (1, 0)$ and $(1, 1)$, the interpolation formula simplifies to

$$f(x, y) = f(0, 0)(1 - x)(1 - y) + f(1, 0)x(1 - y) + f(0, 1)(1 - x)y + f(1, 1)xy \tag{7.1.3}$$

or in matrix form

$$f(x,y) = \begin{pmatrix} 1-x & x \end{pmatrix} \begin{pmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{pmatrix} \begin{pmatrix} 1-y \\ y \end{pmatrix}.$$

This interpolation method does not give highly accurate magnetic field distributions near the surfaces of superconducting layers, but are used as the basic testing method when subdividing the image into quadtree structures, see Chapter 8

### 7.1.4 Bicubic Interpolation

This is a high-performance implementation of pixel interpolation. This algorithm interpolates the nearest sixteen mapped source pixels [31].

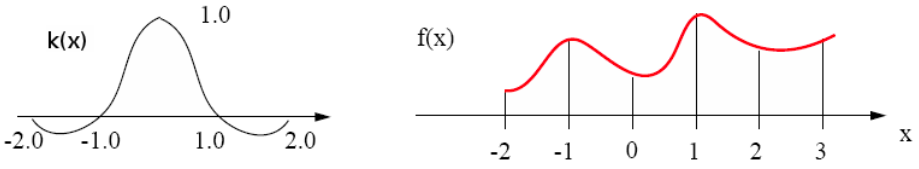
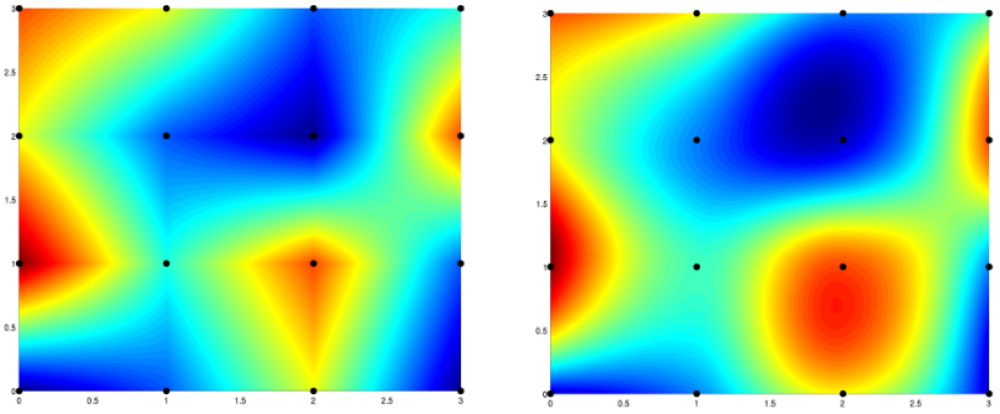



Figure 7.3: Bicubic Kernel

Fig. 7.4 shows the difference between bilinear and bicubic interpolation using the MATLAB functions.



(a) Bilinear Interpolation (b) Bicubic Interpolation

Figure 7.4: MATLAB Interpolation Algorithms

The implementation uses the following cubic spline function as a separable filter to perform a convolution interpolation:

$$f(x; -1 \leq a < 0) = \begin{cases} (a+2)|x|^3 - (a+3)x^2 + 1 & \text{for } 0 \leq |x| \leq 1 \\ a|x|^3 - 5ax^2 + 8a|x| - 4a & \text{for } 1 < |x| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

The $a$ constant parameter of the cubic spline controls the depth of the negative lobes of the interpolation function. In the most common case the $a$ value is fixed to $a = -1/2$, then in matrix form we have

$$f(x) = \frac{1}{2}\begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} a_{-1} \\ a_0 \\ a_1 \\ a_2 \end{pmatrix}.$$

The resulting function profile can be seen in Fig. 7.3. For color mapping, we have extrapolated the values of the left image in Fig. 7.3 to be limited in the $0 \leftrightarrow 255$ interval.

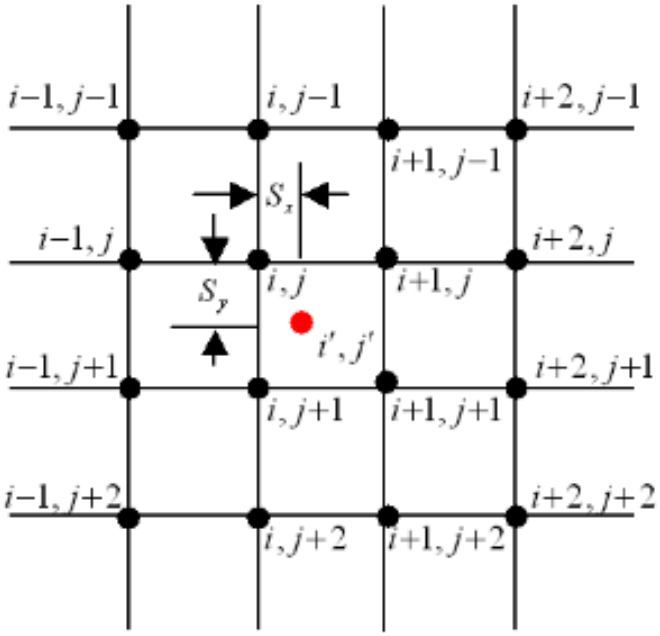



Figure 7.5: Magnetic field value is calculated a the red dot, with the reference point of the bicubic square at node $(i, j)$

Due to the negative lobes of the bicubic spline interpolation function, this algorithm generates undershoot artifacts. This problem can be solved by implementing a clamping mechanism. To do this we use the following linear value

shifting algorithm on each pixel **rgb** color value:

$$y = \frac{255}{x_{max} - x_{min}}(x - x_{min}) \tag{7.1.4}$$

where $x$ is either **r**, **g** or **b** and $y$ is the new color value. The conventional bicubic interpolation needs an up-sampling distance $S$ to estimate the unknown pixels for the interpolation process. At the position $(i', j')$, which is shown in Fig. 7.5, the bicubic interpolation calculates the interpolated pixel as

$$f_{i',j'} = \begin{pmatrix} W_{-1}(S_y) & W_0(S_y) & W_1(S_y) & W_2(S_y) \end{pmatrix} \bullet \begin{pmatrix} f_{i-1,j-1} & f_{i,j-1} & f_{i+1,j-1} & f_{i+2,j-1} \\ f_{i-1,j} & f_{i,j} & f_{i+1,j} & f_{i+2,j} \\ f_{i-1,j+1} & f_{i,j+1} & f_{i+1,j+1} & f_{i+2,j+1} \\ f_{i-1,j+2} & f_{i,j+2} & f_{i+1,j+2} & f_{i+2,j+2} \end{pmatrix} \bullet \begin{pmatrix} W_{-1}(S_x) \\ W_0(S_x) \\ W_1(S_x) \\ W_2(S_x) \end{pmatrix}.$$

where $S_x = i' - i$ and $S_y = j' - j$. The weights are given as:

$$W_{-1}(S) = \frac{-S^3 + 2S^2 - S}{2}, \qquad W_1(S) = \frac{-3S^3 + 4S^2 + S}{2}$$
$$W_0(S) = \frac{3S^3 - 5S^2 + 2}{2}, \qquad W_2(S) = \frac{S^3 - S^2}{2}$$

The algorithm is used to model the transition between two original pixels, and it describes a series of polynomials with a given degree. The higher the degree, the higher the variations in the curve. A bicubic curve of degree one is formed by a series of straight lines, and therefore corresponds to a bilinear kernel. A bicubic curve of degree two is composed of a series of parabolic curves and B-spline of degree three is composed of cubic curves.

### 7.1.5 Biot-Savart Bicubic

Bicubic interpolation is used to improve the speed of Biot-Savart by reducing the amount of calculation points and then interpolating between those points.

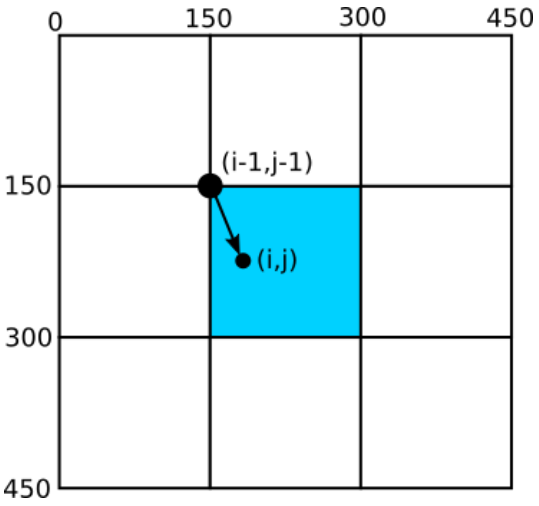
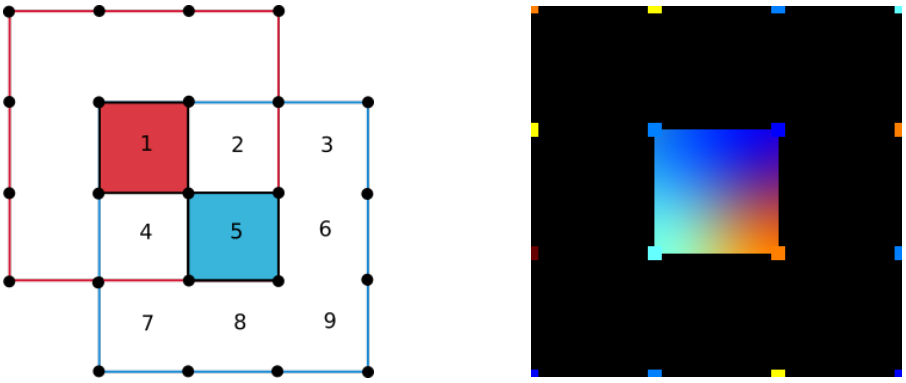



Figure 7.6: Interpolation is done in the blue colored area

The speed factor improves quadraticly, for instance if the grid size is $500 \times 500$ and the magnetic field is calculated at every tenth point, then the new grid becomes $50 \times 50$. Which reduces the calculation points with a factor of $N^2$. Fig. 7.7 shows the sixteen points that are taken into account to interpolate inside the center cube.




(a) 16 Virtual Points (b) Interpolation Example

Figure 7.7: Bicubic Interpolation

To apply interpolation to the field calculations we first divide the original $\Omega_O$ grid space up into any amount of equally sized square, which create a new grid $\Omega_N$. Let the point at which we want to calculate the magnetic field be defined as $(i', j')$. Then find the nearest grid point $(i, j)$ in $\Omega_N$, that holds

$i' - i > 0$ and $j' - j > 0$ true. In other words, the nearest new grid point that is top-left of the observation point. Now that we have the square $\Omega_S$ in which the point $(i', j')$ lies, we can create the 16 virtual nodes centered around the square $\Omega_S$ to apply the bicubic interpolation.

## 7.2 Adaptive Algorithms for Image Interpolation

Since the conventional bicubic interpolation has blurring problems, one modified method was proposed in [32] using the image local asymmetry features, and another modified method was proposed in [23] using the image local gradient features. We combined these two features to improve the clarity of magnetic field lines by overcoming these blurring problems.

Adaptive interpolation is proposed based on logics of local structure of the image and intensity variations that are indistinguishable by human eye. The approach proposed in [32] uses local gradient information to adapt the bilinear and bicubic interpolation algorithms to yield better perceptual image quality and improved image metrics. In this method, the weights of the input image pixels which in conventional interpolation algorithms are functions of distance only are modified by dividing the normalized local gradients to yield a better interpolation estimation when the gradient changes abruptly in edge regions. This is important for improving magnetic field accuracy near the surface of the superconducting layers. Intelligent observation grid generation also plays an important part in improving magnetic field curve accuracy, as explained in Chapter 8.

The local asymmetry feature based bicubic method uses the modified interpolation value $f^{\mathbf{A}}_{i',j'}$, to replace the conventional interpolation value $f_{i',j'}$ as follow

$$f^{\mathbf{A}}_{i',j'} = \begin{pmatrix} W_{-1}(S'_y) & W_0(S'_y) & W_1(S'_y) & W_2(S'_y) \end{pmatrix} \bullet \begin{pmatrix} f_{i-1,j-1} & f_{i,j-1} & f_{i+1,j-1} & f_{i+2,j-1} \\ f_{i-1,j} & f_{i,j} & f_{i+1,j} & f_{i+2,j} \\ f_{i-1,j+1} & f_{i,j+1} & f_{i+1,j+1} & f_{i+2,j+1} \\ f_{i-1,j+2} & f_{i,j+2} & f_{i+1,j+2} & f_{i+2,j+2} \end{pmatrix} \bullet \begin{pmatrix} W_{-1}(S'_x) \\ W_0(S'_x) \\ W_1(S'_x) \\ W_2(S'_x) \end{pmatrix}.$$

where

$$S'_y = \frac{S'_{y1} + (S'_{y2} - S'_{y1})S'_{x1}}{1 - (S'_{y2} - S'_{y1})(S'_{x2} - S'_{x1})}$$

$$S'_x = \frac{S'_{x1} + (S'_{x2} - S'_{x1})S'_{y1}}{1 - (S'_{y2} - S'_{y1})(S'_{x2} - S'_{x1})}$$

and

$$S'_{x1} = S_x - kA_{x1}S_x(1-S_x), \qquad S'_{y1} = S_y - kA_{y1}S_y(1-S_y)$$
$$S'_{x2} = S_x - kA_{x2}S_x(1-S_x), \qquad S'_{y2} = S_y - kA_{y2}S_y(1-S_y)$$

The parameters $A_{x1}$, $A_{x2}$, $A_{y1}$ and $A_{y2}$ are the local asymmetry features defined by

$$A_{x1} = \frac{|f_{i+1,j} - f_{i-1,j}| - |f_{i+2,j} - f_{i,j}|}{L-1}$$
$$A_{x2} = \frac{|f_{i+1,j+1} - f_{i-1,j+1}| - |f_{i+2,j+1} - f_{i,j+1}|}{L-1}$$
$$A_{y1} = \frac{|f_{i,j+1} - f_{i,j-1}| - |f_{i,j+2} - f_{i,j}|}{L-1}$$
$$A_{y2} = \frac{|f_{i+1,j+1} - f_{i+1,j-1}| - |f_{i+1,j+2} - f_{i+1,j}|}{L-1}$$

### 7.2.1 Local Gradient Bicubic

In [33], using the local gradient features around the interpolation pixel $f_{i',j'}$ four local gradient weights are proposed. The four local gradient weights $H_l$, $H_r$, $V_u$ and $V_l$ are generated from the four pixel masks around the interpolated pixel. The weights are defined as

$$H_l = \frac{1}{\sqrt{1+\alpha(|f_{i,j} - f_{i-1,j}| + |f_{i,j+1} - f_{i-1,j+1}|)}}$$
$$H_r = \frac{1}{\sqrt{1+\alpha(|f_{i+1,j} - f_{i+2,j}| + |f_{i+1,j+1} - f_{i+2,j+1}|)}}$$
$$V_u = \frac{1}{\sqrt{1+\alpha(|f_{i,j} - f_{i,j-1}| + |f_{i+1,j} - f_{i+1,j-1}|)}}$$
$$V_l = \frac{1}{\sqrt{1+\alpha(|f_{i,j+1} - f_{i,j+2}| + |f_{i+1,j+1} - f_{i+1,j+2}|)}}$$

where the parameter $\alpha$ is in the range of $[0, 1]$. Using the local gradient weights we get eight new interpolation weights $W_n^G$ to replace the conventional interpolation weights $W_n$ in the bicubic interpolation. Then, the local gradient feature base bicubic method is given as

$$f^G_{i',j'} = \begin{pmatrix} W^G_{-1}(S_y) & W^G_0(S_y) & W^G_1(S_y) & W^G_2(S_y) \end{pmatrix} \bullet \begin{pmatrix} f_{i-1,j-1} & f_{i,j-1} & f_{i+1,j-1} & f_{i+2,j-1} \\ f_{i-1,j} & f_{i,j} & f_{i+1,j} & f_{i+2,j} \\ f_{i-1,j+1} & f_{i,j+1} & f_{i+1,j+1} & f_{i+2,j+1} \\ f_{i-1,j+2} & f_{i,j+2} & f_{i+1,j+2} & f_{i+2,j+2} \end{pmatrix} \bullet \begin{pmatrix} W^G_{-1}(S_x) \\ W^G_0(S_x) \\ W^G_1(S_x) \\ W^G_2(S_x) \end{pmatrix}.$$

where the weights $W_n^G(S_y)$ are given as

$$W_{-1}^G(S_y) = \frac{W_{-1}(S_y)}{DV(S_y)}, \qquad W_1^G(S_y) = \frac{V_l W_1(S_y)}{DV(S_y)}$$
$$W_0^G(S_y) = \frac{V_u W_0(S_y)}{DV(S_y)}, \qquad W_2^G(S_y) = \frac{W_2(S_y)}{DV(S_y)}$$

and the weights $W_n^G(S_x)$ are given as

$$W_{-1}^G(S_x) = \frac{W_{-1}(S_x)}{DH(S_x)}, \qquad W_1^G(S_x) = \frac{H_r W_1(S_x)}{DH(S_x)}$$
$$W_0^G(S_x) = \frac{H_l W_0(S_x)}{DH(S_x)}, \qquad W_2^G(S_x) = \frac{W_2(S_x)}{DH(S_x)}$$

here

$$DV(S_y) = W_{-1}(S_y) + V_u W_0(S_y) + V_l W_1(S_y) + W_2(S_y)$$
$$DV(S_x) = W_{-1}(S_x) + H_l W_0(S_x) + H_r W_1(S_x) + W_2(S_x)$$

### 7.2.2 High Accuracy Bicubic Interpolation

To adopt both the local asymmetry feature and the local gradient feature, we give the proposed modified interpolation weights $W_n^p$ as

$$W_{-1}^p(S_y) = W_{-1}^G(S'_y), \qquad W_1^p(S_y) = W_1^G(S'_y)$$
$$W_0^p(S_y) = W_0^G(S'_y), \qquad W_2^p(S_y) = W_2^G(S'_y)$$

and

$$W_{-1}^p(S_x) = W_{-1}^G(S'_x), \qquad W_1^p(S_x) = W_1^G(S'_x)$$
$$W_0^p(S_x) = W_0^G(S'_x), \qquad W_2^p(S_x) = W_2^G(S'_x)$$

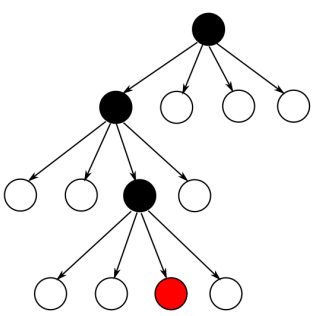
in which $S'_x$, $S'_y$ are based on the image local asymmetry features, and $W_n^G$ are based on the image local gradient features. Then, the proposed bicubic interpolation can be written as

$$f_{i',j'}^p = \begin{pmatrix} W_{-1}^p(S_y) & W_0^p(S_y) & W_1^p(S_y) & W_2^p(S_y) \end{pmatrix} \bullet \begin{pmatrix} f_{i-1,j-1} & f_{i,j-1} & f_{i+1,j-1} & f_{i+2,j-1} \\ f_{i-1,j} & f_{i,j} & f_{i+1,j} & f_{i+2,j} \\ f_{i-1,j+1} & f_{i,j+1} & f_{i+1,j+1} & f_{i+2,j+1} \\ f_{i-1,j+2} & f_{i,j+2} & f_{i+1,j+2} & f_{i+2,j+2} \end{pmatrix} \bullet \begin{pmatrix} W_{-1}^p(S_x) \\ W_0^p(S_x) \\ W_1^p(S_x) \\ W_2^p(S_x) \end{pmatrix}.$$

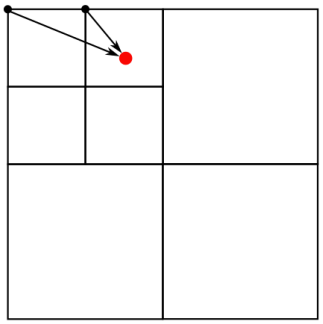
# Chapter 8

# Quad Tree

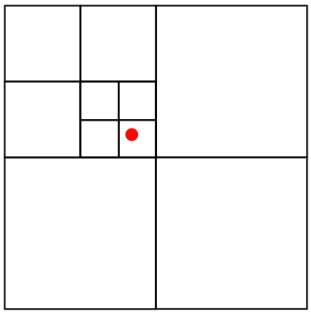
Quadtree encoding is used to represent a two-dimensional region, by dividing the region into quadrants. To enable recursive tree traversal in both directions, from parent to children or otherwise, each cell has a pointer to its parent cell and to its four children. The quadtree grid can be represented by a



(a) Quadtree Data Structure



(b) Quadtree Spacial Position



(c) Quadtree $3^{rd}$ Level Subdivision

Figure 8.1: Quadtree Breakdown

directed graph with tree-like structure. To any quadtree cell system, a tree-like data structure can be attached, and usual tree-traversal algorithms can

be used for data handling. The use of node pointers eliminates the need to repeatedly store an integer counter for the purpose of indexing in generating a list-based computational grid from the quadtree. Subdividing the observation grid using quadtree spatial subdivision will improve calculation time of magnetic fields [34], [35], since we loop recursively through the grid. Also since the quadtree algorithm is based on object detection we can converge to the boundaries of the layer polygons. This will improve the accuracy calculations near the surface of the superconducting layers [36]. Bicubic interpolation will also be improved since the cubic blocks become smaller near the surface of the layers.

## 8.1 Quadtree Grid Generation

Let $\Omega_0$ be an initial square and let $S_0$ be a points contained in $\Omega_0$. We call this initial pair $(\Omega_0, S_0)$, the 0th level of subdivision. The quadtree algorithm is a recursively defined procedure which operates on a pair $(\Omega, S)$. First, divide $\Omega_0$ into four equally sized subsquares $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ and define the subsets $S_1, S_2, S_3, S_4$ by being the points inside each corresponding subspace, then increase the level by one. Do this subdivision recursively for each of the pairs $(\Omega_1, S_1)$, $(\Omega_2, S_2)$, $(\Omega_3, S_3)$ and $(\Omega_4, S_4)$. The algorithm is finished when the region is divided into the minimum sized squares $L_0$, which is user defined.

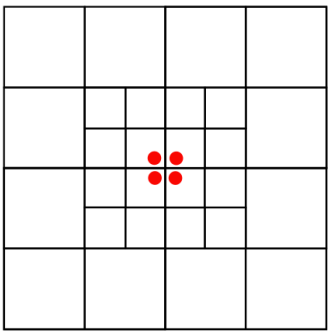
### 8.1.1 Computation Time

The computational cost of the algorithm is very low. A rough estimate for the number of the necessary operations is $O(N \times L)$, where $N$ is the number of points $S$ and $L$ is the level of subdivision. However, since it is a recursive function, the maximum level of subdivision is proportional to the number $logN$. Therefore, the cost of the grid generation is $O(N \cdot logN)$ only.

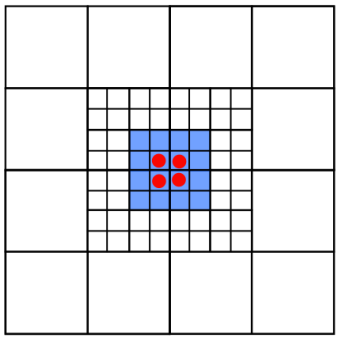
## 8.2 Layer Boundary Detection

The observation grid, $L$, which represents the magnetic field's spatial domain, is subdivided recursively until a user specified minimum grid size is obtained. The object points, $S_i$, to which the quadtree converges is the boundary points of each superconducting layer [37], [38]. These boundary points construct the polygon which represent the layer as seen from the cross-sectional view. The polygons are constructed using the Clippers C++ library [39].

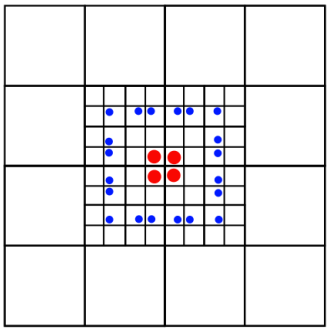
The sliced segments in the sliced layers are found and saved in a new array. The polygon of the cross-sectional view of each segment is found, and the unify method in the Clippers library are used to construct the larger cross-sectional polygon of the layer. In Fig. 8.2 the space is divided into a quadtree structure



(a) Third Level Subdivision



(b) Near fields



(c) Near fields and virtual nodes

Figure 8.2: Generating Near and Far Fields

for four individual points of interest. In the final tool these points will represent the boundary coordinates of a superconducting layer. Each individual quadtree block or node will be used as the basis for the bicubic interpolation algorithm to determine the magnetic fields inside the space bounded by the quadtree block dimensions [40]. In some cases this may become very inaccurate, since the block sizes increases and decreases quadraticaly. To improve the accuracy of the system, we divide the space up into near and far fields. This is done by expanding the level of subdivision for the previous $n$ levels by some factor. For instance, the level of subdivision in Fig. 8.2a is three. If we subdivide it by two more levels and then subdivide the previous nodes by one more level, we get the image Fig. 8.2b. In this figure the area in blue is the original minimum sized squares, and from there on the previous parent nodes are subdivided using virtual nodes. The near fields are defined as the area covered by a user defined sized square and smaller.

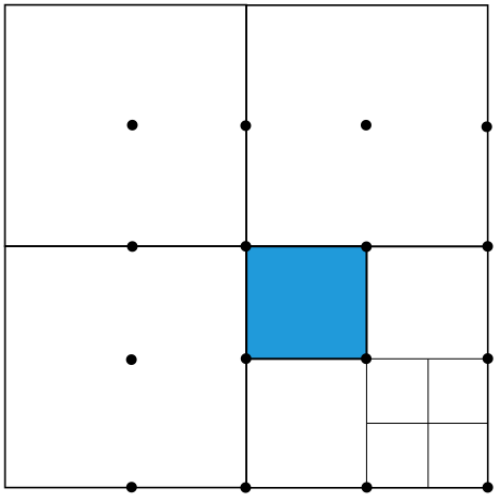


Figure 8.3: Biot-Savart points used to calculate magnetic fields inside square

We can do this extra subdivision by creating virtual nodes that the quadtree algorithm can detect [41]. Fig. 8.3 shows the points at which the magnetic field vectors will be calculated. For each node in the quadtree create 16 grid points symmetrically around the node and apply Biot-Savart to each of these 16 points. The bicubic algorithm is then applied to calculate the magnetic field inside the square, colored in blue. This is done for every square inside the quadtree. It is important to note not to interpolate when the current square has smaller squares inside it, since these smaller squares themselves will be interpolated with a much higher accuracy.

## 8.3 Virtual Quadtree

Create a second quadtree, this time adding virtual points to force the space to be divided. Toggle a flag in the node struct. This flag decides whether a point will be plotted or taken into account when calculating the magnetic fields. A virtual node will not be taken into account when plotting or calculations are performed. Eight virtual nodes are added to each real node, as shown in Fig. 8.4. These virtual nodes will subdivide the parent node and other neighbouring parent nodes into the smaller squares. A variable is set to represent the distance these virtual nodes are from the real node in terms of grid units. By default the variable is one, but if set to two then the parent's parent (grandparent) will be subdivided into the parent's size squares. Every time we add virtual nodes, we have to recreate the quadtree, since it is to difficult to go back and manually change the quadtree structure, by adding and removing

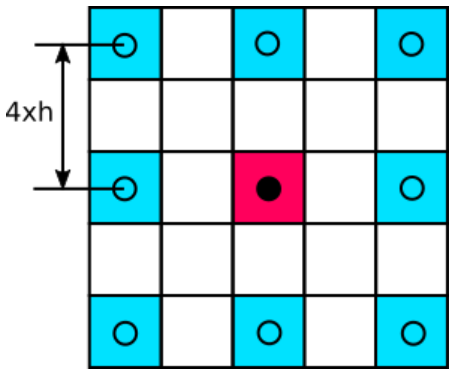



Figure 8.4: Virtual nodes around a real node

new nodes. Fig. 8.4 shows in blue the virtual nodes created over one real node with a depth level of one.

### 8.3.1 Fast Multipole Method

It might be possible to improve the speed calculations of magnetic fields using the fast multipole method (FMM), but this has not been studied in great depth [42]. Greengard and Rokhlin [43] stated that the FMM can in principle be used for efficient computation of Biot-Savart. In the FMM algorithm, field computation is split into a near-field part $\{\mathbf{B}_{near}\}$ and a far-field part $\{\mathbf{B}_{far}\}$, which gives

$$\{\mathbf{B}_J\} = \{\mathbf{B}_{near}\} + \{\mathbf{B}_{far}\} \tag{8.3.1}$$

The division of element interactions into a near-fields and a far-field part has be discussed in this chapter. This FMM method was not implemented in the reported tool, but are only discussed for future references.

# Chapter 9

# Visualization Results

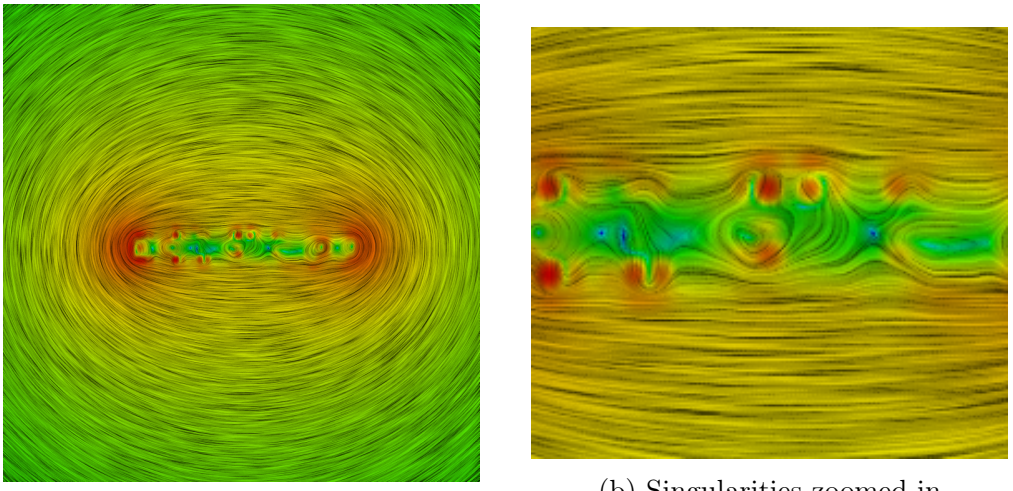
The chapter is divided into three sections, the first is to show the improvements and enhancements made to calculating and visualizing the magnetic fields. The user can use different plotting methods for printing purposes and other methods for electronic formats. The second part shows practical examples and results obtained by the toolkit. And the third part discusses the possible future work that can be done to create an auto-hole generator. Basic code was written for geometric processing of holes around superconducting layers and some of the results are shown.

## 9.1 Magnetic Fields

We present the results of the magnetic fields starting from the first versions of the program, and ending at the final tool results. The development of the program was divided into four sub-projects:

- Calculate the magnetic fields for both FFH and TH, by focusing on numerical accuracy and implementing different visualization methods.
- Improve the speed of the calculations.
- Improve visualization algorithms and techniques, by focusing on speed and visualization clarity.

In this chapter the current distributions though each segment used to calculate the magnetic fields was calculated by the FFH engine. The magnetic fields calculated with trapped flux was done using the current distribution results obtained from the TH engine. Fig. 9.1 shows the magnitude of the magnetic fields around a simple current carrying conductor in free space. Inside the superconducting layer the points taken by the Gaussian quadrature algorithm causes a magnitude maximum, which affects the rest of the field's magnitude values. To compensate for this, Biot-Savart was not calculated for



(a) Layer with Singularities

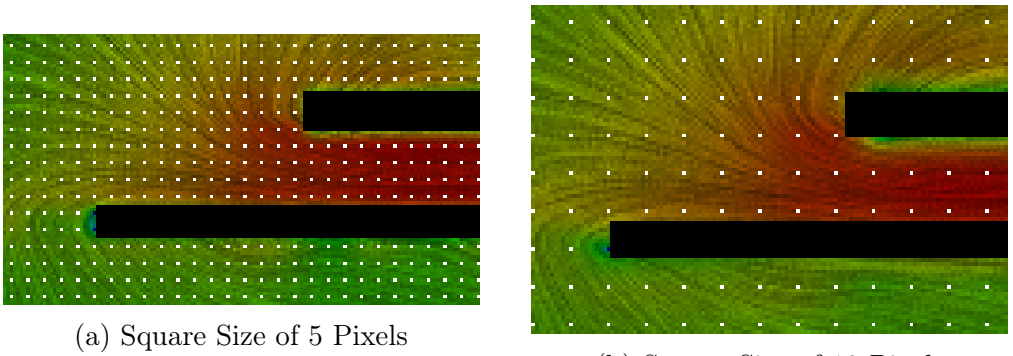
(b) Singularities zoomed in

Figure 9.1: Singularity in Biot-Savart using TetraHenry

points inside the layer. This also solves the problems concerning singularities in the quadrature algorithm, Fig. 9.1b shows these maximum points.

### 9.1.1 Bicubic Speed Improvement

The bicubic interpolation algorithm where implemented to improve the speed of calculating Biot-Savart's law over the spacial domain. The example shown in this section was run over a $500 \times 500$ pixel image. As can be seen in Fig. 9.2 the larger the squares of the bicubic algorithm, the more inaccurate the field calculations near the surface of the superconducting layers. This is only the



(a) Square Size of 5 Pixels

(b) Square Size of 10 Pixels

Figure 9.2: Magnetic Fields are calculated at the dot positions using Biot-Savart

case if we ignore the magnetic fields inside the layers. When the FFH engine is used the magnetic fields inside the layers are taken into account, which gives a smoother result near the layers surfaces. The default bicubic square

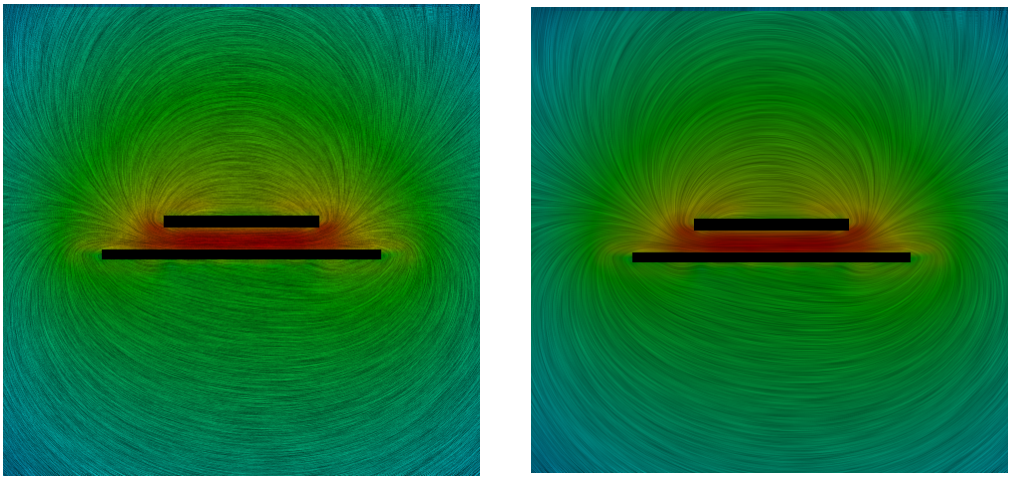
size is 10 pixels, and comparing the two images in Fig. 9.2 the results looks acceptable. Magnetic fields from the Biot-Savart algorithm is only calculated at every $n$th pixel, where $n$ is the bicubic square size given in the left column in the table below. The bicubic interpolation algorithm calculates the magnetic field values inside each square as described in Chapter 7. The table below shows the simulation times in seconds for three different bicubic square sizes.

| Square (Pixels) | Biot-Savart (sec) | Bicubic (sec) |
|---|---:|---:|
| 2 | 793.653 | 58.193 |
| 5 | 117.884 | 10.621 |
| 10 | 31.572 | 2.349 |

From Fig. 9.2 the bicubic high accuracy algorithm gives very similar results to that of the normal bicubic algorithm with a smaller square size. To improve the accuracy near the layer surfaces the spacial domain can be subdivided using quadtree algorithms.

### 9.1.2 Enhanced LIC

Fig. 9.3a shows the magnetic fields using just the LIC method in its default form, with no enhancement or speed improvement. Fig. 9.3b shows the double



(a) Original LIC Algorithm (b) Double LIC Algorithm

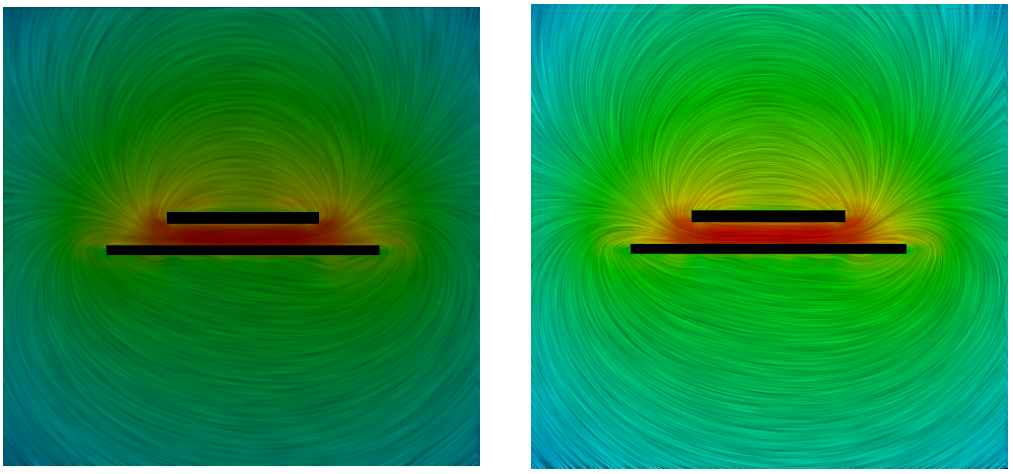
Figure 9.3: Second convolution is applied to original LIC

LIC algorithm where convolution is applied twice. Comparing the two results the enhancement smooths out the magnetic field lines. This method also improves edge detection, which causes separation of the field lines. From here on the double LIC method will be used as the default. Even though the spreading of the field lines can easily be observed, in some cases enhancing the near fields plays an important role. Applying different filters and edge

detection methods can improve the near field visualization. These methods will also point out smaller field convergence points in the magnetic field.

### 9.1.3 Low- and High-pass Filters

The low-pass filter blurs the image and has no use on its own, but will play an important role in later methods, such as the DWT algorithm and more so in the sharpening filter. The high-pass filter enhances the color visualization of the



(a) Low-pass Filter (b) High-pass Filter

Figure 9.4: Filters applied to double LIC

magnitude and makes it easier to distinguish between changes in magnitude over the spacial domain. In Fig. 9.4 a low- and high-pass filter is applied to the double LIC method. The following $3 \times 3$ low-pass filter where used:

$$\begin{bmatrix} 0.0 & 0.2 & 0.0 \\ 0.2 & 0.2 & 0.2 \\ 0.0 & 0.2 & 0.0 \end{bmatrix} \tag{9.1.1}$$

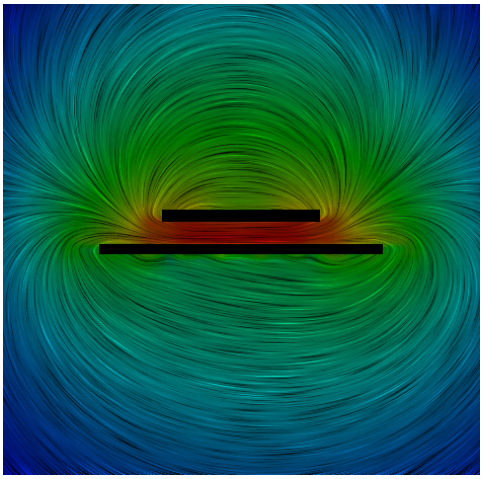
and high-pass filter:

$$\begin{bmatrix} 0.0 & 0.5 & 0.0 \\ 0.5 & 1.5 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} \tag{9.1.2}$$

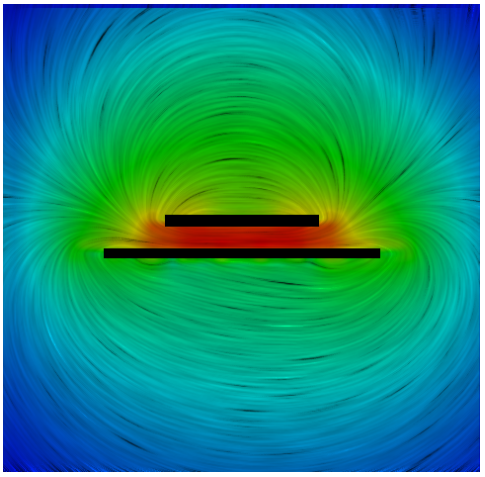
These low-pass and high-pass filters are also used in the DWT algorithm. The DWT method was not implemented into the final program, but can become significant in the future for upscaling and downscaling when zooming in and out to keep image quality.

### 9.1.4 Histogram Filter

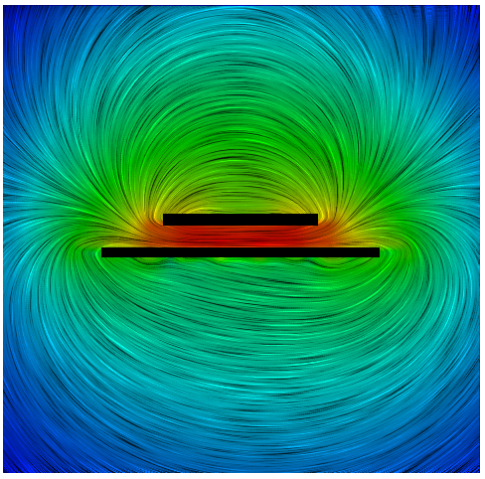
Magnetic field lines can be enhanced by filtering the higher and lower values of the color spectrum, using a histogram filter. This is explained in Chapter 6. Fig. 9.5 shows a histogram filter applied over the spectrum between 150-200



(a) Normal LIC with Histogram Filter



(b) Low-pass LIC with Histogram Filter



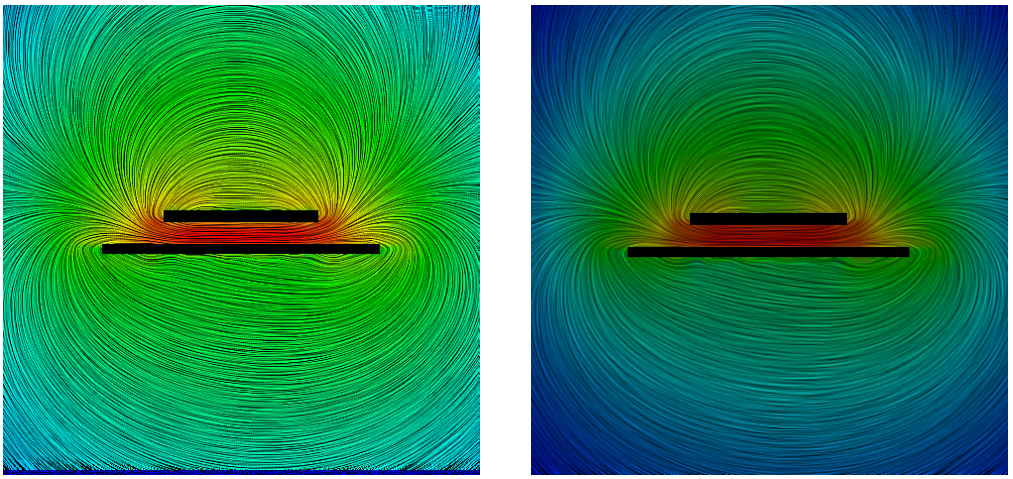
(c) High-pass LIC with Histogram Filter

Figure 9.5: Histogram filter

**rgb** values. Convolution is done with low- and high-pass filters to compare with the results in the previous section. Fig. 9.5c shows the default magnetic field visualization used in the final version of the program.

### 9.1.5 Sharpening Filter

Applying a sharpening filter to the LIC method, over enhances the magnitude of the magnetic fields. The magnetic field lines looks more continues over the spacial dimension, Fig. 9.6. By applying a low-pass filter we can decrease the



(a) Sharpen Filter (b) Sharpen and Low-pass Filter

Figure 9.6: Sharpen filter applied to Double LIC

over saturation as shown in Fig. 9.6b. The sharpening filter used:

$$\begin{bmatrix} -1.0 & -1.0 & -1.0 \\ -1.0 & 9.0 & -1.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix} \tag{9.1.3}$$

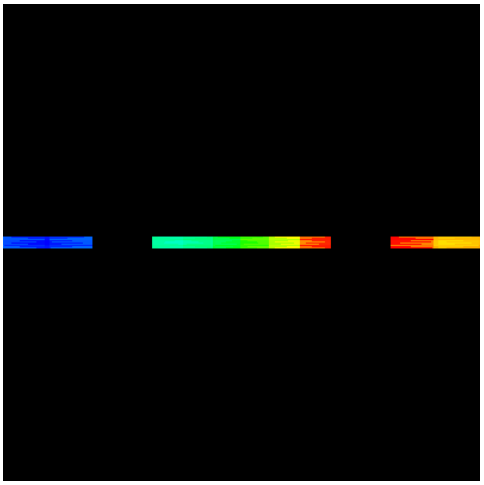
Two other edge detection filters where applied for experimentation purposes: The first is that of the DWT method explained in Chapter 6, and the other is the following edge detection filter:

$$\begin{bmatrix} -1.0 & -1.0 & -1.0 \\ -1.0 & 8.0 & -1.0 \\ -1.0 & -1.0 & -1.0 \end{bmatrix} \tag{9.1.4}$$

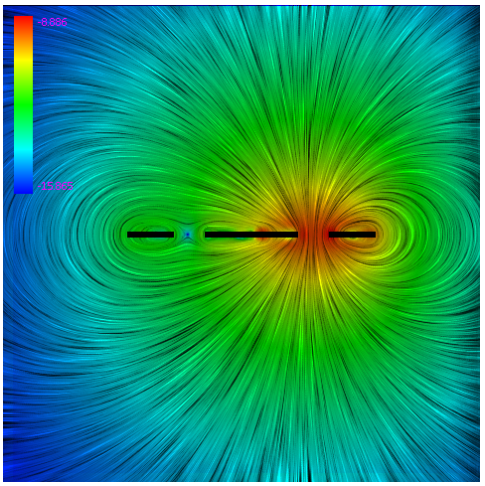
The edge filter under saturates the image and makes it very difficult to see the individual field lines. The DWT method improved the image quality by a small fraction. The above two method were not incorporated into the final version of the tool chain. The DWT method's strong point is in image upscaling and downscaling and does not enhance the edges enough if the image dimensions stays the same.

## 9.2 Magnetic Fields caused by Trapped Flux

One of the most important applications of MagnetEx is to visualize the magnetic fields produced by trapped flux. Fig. 9.7 shows two holes in the GP layer. The legend showing the magnetic magnitude in log scale are given on the top left side of the image. The user can choose to show or hide the legend. The left hole has no fluxon trapped and the right hole has one fluxon



(a) Current Distribution with Trapped Flux



(b) Magnetic Fields of Trapped Flux

Figure 9.7: Two GP holes with one trapped flux

trapped. The image on the left shows the current density distribution and the one on the right shows the magnetic field distribution. The current distribution visualized in this image is only an approximation, since determining the exact current at each point inside a tetrahedron is very time expensive. The reason being, that we must first find the tetrahedron in which the point lays. Thereafter, a set of current determining algorithms must be applied to detect the resultant current at that point. The basic underlining algorithmic methods are described in Chapters 2 and 4. The exact current distribution can be calculated by MagnetEx, but here the more raw version is shown just to give an idea of how the current is spread over the cross-sectional view of the layers.

Future investigations will involve determining the maximum magnetic field magnitude caused by the current in one layer allowed to penetrate a different layer. This will result in a coupling factor between the two layers. Factors that might also have to be taken into account is the direction with which the field lines penetrate the layer surfaces.

### 9.2.1 Practical Example

The problem considered here are one setup by Y. Yamanashi and his student H. Imai and was presented at the ISEC conference in Nagoya, Japan in July 2015 [44]. The stripline has a thickness of 400 $nm$, width of 3 $\mu m$ and length of 20 $\mu m$. The hole has a width of 900 $nm$ and a length of 5 $\mu m$. The separation between the stripline and the hole is 1 $\mu m$. Generally superconducting

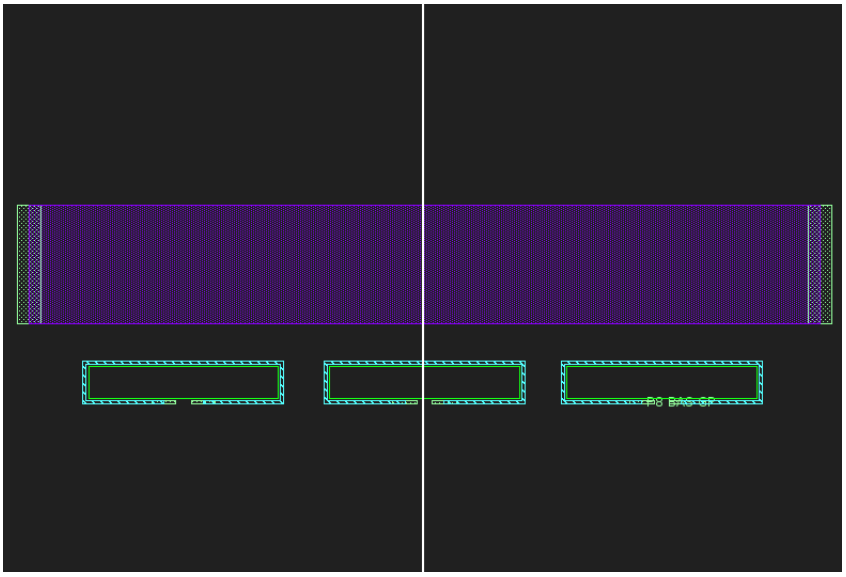


Figure 9.8: GDS View

circuits are sensitive to internal magnetic fields produced by the bias current, external magnetic fields and magnetic fields caused by trapped flux. The aim

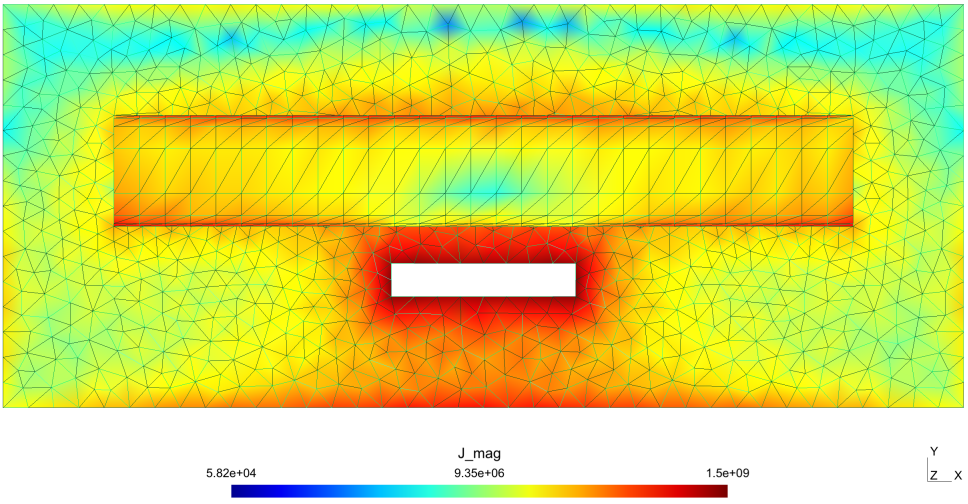



Figure 9.9: Flux current stored clockwise

of this problem setup is to investigate the influence of trapped flux in moats on superconducting circuit operations. The GDS view of the example is shown in Fig. 9.8 using a GDS viewer program created by the author of this thesis. The tool is written in the Python programming language and are used to select the cross-sectional slice that will be made by MagnetEx. The slice can either be

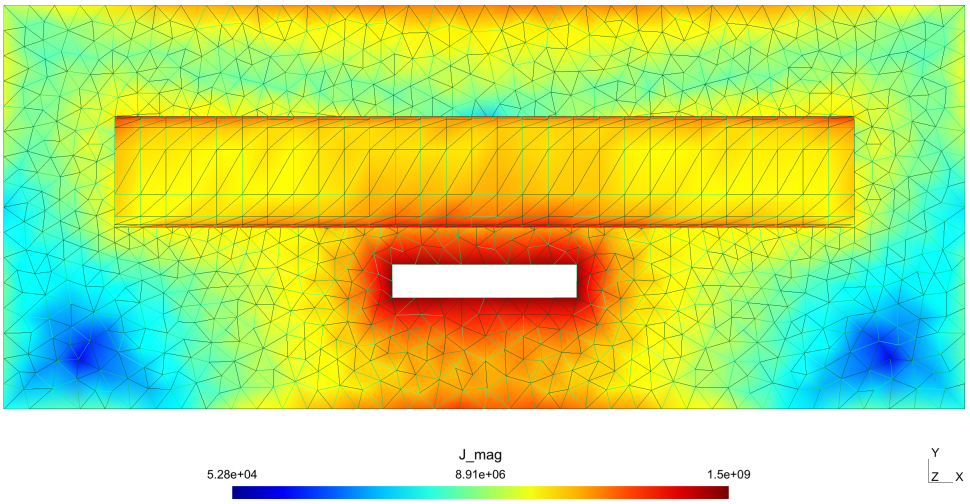
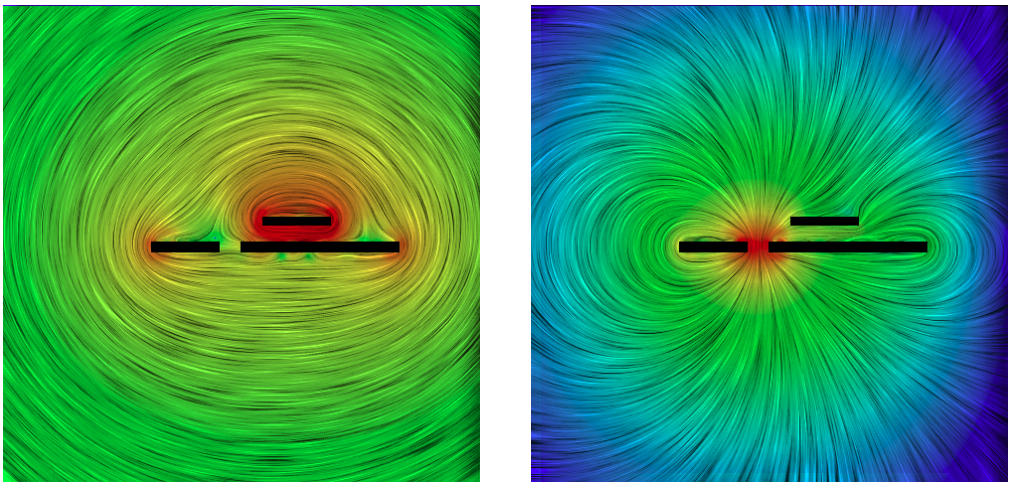



Figure 9.10: Flux current stored counterclockwise

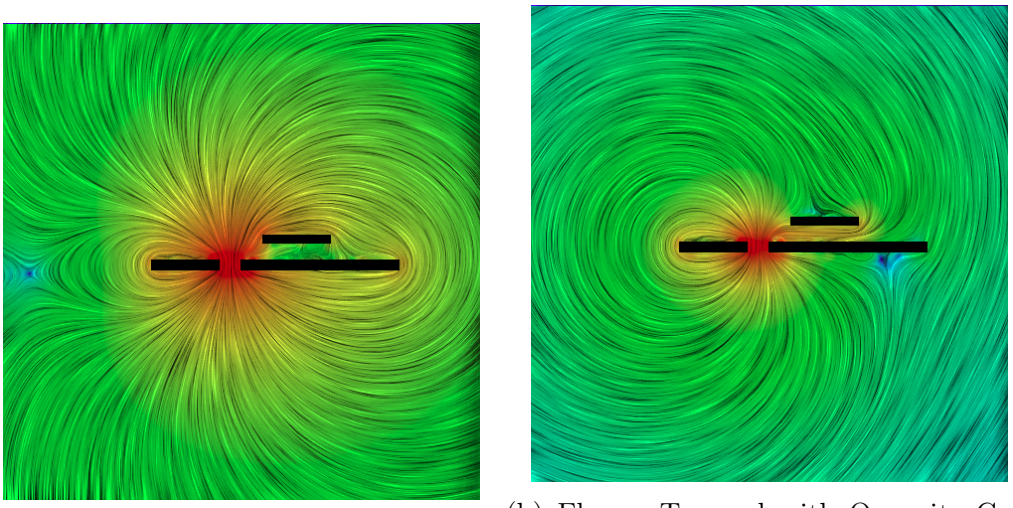
vertical or horizontal. In this example a vertical slice is taken as presented by the vertical white line shown in the figure. Using the TH engine the example was setup to investigate the effect of one trapped fluxon in the second hole positioned next to the center of the strip line. The current distribution from the top view are shown in Fig. 9.9 and Fig. 9.10. The difference between the



(a) Strip Line Excited (b) Flux Trapped in GP Hole

Figure 9.11: Excited currents in this example

two images is that the current of stored flux is reversed. The superposition of the flux current and the strip line current results in a maxima and minima located at different positions. The current minima are represented by the blue colored areas in Fig. 9.9 and Fig. 9.10. In this example the current in the strip line is from right to left. In Fig. 9.9 we see that the stored flux current decreases the current in the strip line. The results in Fig. 9.11 and Fig.



(a) Strip Excited and Fluxon Trapped

(b) Fluxon Trapped with Opposite Current Direction

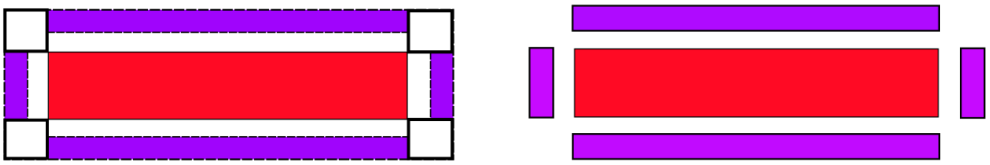
Figure 9.12: Superposition of strip current and trapped flux

9.12 where calculated with MagnetEx using the TetraHenry engine. The fields visualized are broken down into three parts. First visualizing the field with only current going through the strip line. The second part has only a fluxon trapped inside the ground plane hole. And the third part shows the resultant magnetic fields with a fluxon trapped and the strip line excited. The fields Fig. 9.12a corresponds to the current distribution in Fig. 9.9, and the fields in Fig. 9.12b corresponds to the current distribution if Fig. 9.10.

## 9.3 Hole Placement

The behaviour of defect free type II superconductors is well explained by the calculations of Abrikosov [45] and based on the Ginzburg-Landau theory. However, for all practical purposed in single flux quantum electronics calculating and visualizing magnetic fields using Biot-Savart's law will suffice. It is important to visualize the magnetic fields in and around fluxoids, since the consist of a current density and a magnetic field distribution around a small normally conducting core. In the presence of a current density $J$, a force $F = \phi_0 J/2c$ is exerted on a fluxoid and is directed so as to move the fluxoid to a lower magnetic position.

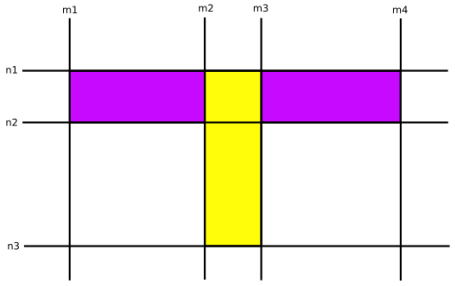
Flux trapping occurs not only in SQUIDs, but may also exist in the ground plane [46], [47]. These trapped flux quanta take on a special form called Abrikosov vortices. Flux trapping happens during the fabrication process. When cooled down the circuit as a whole does not reach 4 $K$ at the same time.



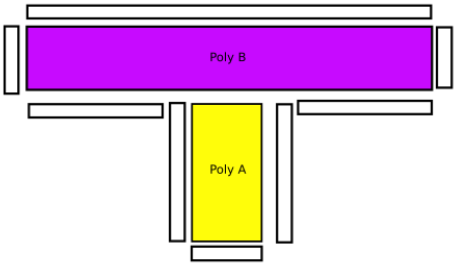
(a) Cutting Four Hole Polygons (b) Four Holes

Figure 9.13: Generating holes around a strip line

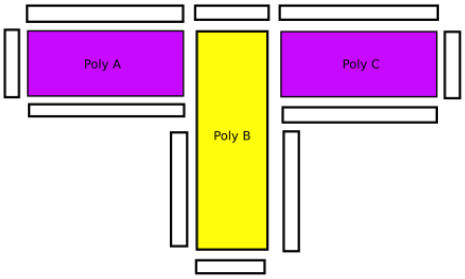
Some parts may cool down faster than others. Thus only certain parts start to become superconducting, while other parts are still non-superconductive. As these different areas grow larger, new once starts to emerge and join with the larger areas. Some normal areas will become surrounded, which will then cause the flux within the normal area to round-off to the nearest integer number of quanta and thus saving a single flux in that area. To overcome this problem of flux trapping, ground plane holes are inserted where ever possible. The problem concerning the placement of holes in a circuit is the fact that the designer adds the holes after the initial circuit layout. Using magnetic fields a auto-hole generator can be designed. The idea is to intelligently place holes around the circuit, then to calculate the effect these holes with trapped fluxons have on circuit operations. The problem becomes more difficult with complex polygon shapes. In Fig. 9.14 the holes can take on a variety of different shapes and sizes and finding the best fit becomes a difficult task. We have to find all the edges of the polygon structure. First place all the vertical holes then all the horizontal holes. These hole placement methods can also be used to place fluxons inside these holes and then see the effect each individual trapped fluxon and the combined fluxons have on the circuit. These hole areas can be




(a) Dividing Structure into Convex Polygons




(b) One possibility of hole placement




(c) Second possibility of hole placement

Figure 9.14: Hole placement of more complex structure

subdivided into smaller areas that represent trapped fluxons like in Fig. 9.15. For this example each fluxons is trapped with a clockwise current rotation. The current direction can also be altered to observe the different effects these holes has on the circuit.

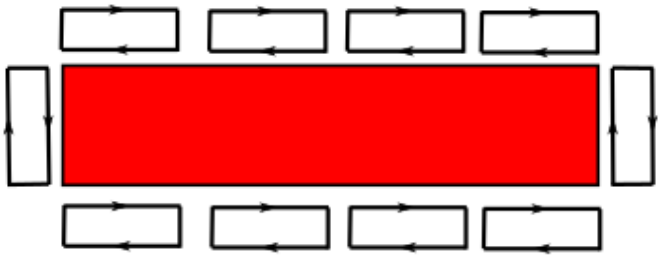


Figure 9.15: Theoretical example of an auto flux trapping generator

The auto-hole generator discussed in this section only briefly outlines the complexity of the problem. The basic idea for future work is to use MagnetEx and geometry manipulations to get acceptable results.

# Chapter 10

# eSFQ Theory

RSFQ logic critically biases its junctions using a resistor in the biasing network. In order to keep the current flowing into the circuit, the biasing voltage $V_b$ must exceed the voltage at the injection point (biasing node) given by

$$(v_i)_{max} = \Phi_0 f \tag{10.0.1}$$

Let $N$ be the amount of times a junction switches inside a gate. If $V_b < (v_i)_{max}$ and $N >> 1$ then a current $I_r$ can start to flow into the biasing network:

$$I_r = f\Phi_0/R \tag{10.0.2}$$

therefore the biasing resistor $R$ has to be made big enough. Power dissipation in SFQ logic can be classified into two catagories:

- Static power dissipation: Power dissipated by biasing networks, which is mostly caused by the biasing resistor.
- Dynamic power dissipation: Power dissipated only by the switching of junctions

The relationship between static- and dynamic power dissipation is [48]

$$\frac{P_s}{P_d} = \frac{V_b}{\Phi_0 f} \tag{10.0.3}$$

From this relation it is clear that in order to decrease the power of RSFQ circuits, the static power has to be decreased drastically. The dynamic power dissipated in each junction is [49]

$$P_D = \Phi_0 I_C f \tag{10.0.4}$$

and from this the conclusion can be made that for ultimate power efficiency, the design of a circuit must use junctions with the lowest possible critical currents. Smaller critical currents means smaller junction areas, which also improves the

space-efficiency of a circuit. In general, junction critical currents are not made less than $100\mu A$, because of random noise and sensitivity to magnetic fields. Also, the smaller the critical current the larger the width of the SFQ pulse becomes, resulting in a decrease of the circuit working frequency. Therefore, a fine balance must be found between power efficiency and clock speed. The same logical thinking can be derived from the following equation, which gives the voltage produced by the switching of a junction [50]

$$V_J(t) = \frac{2I_cR_s}{1+\omega_c^2(t-t_s)^2} \tag{10.0.5}$$

where $R_s$ is the shunt resistance, $t_S$ is the switching time and $\omega_c$ is the characteristic angular frequency as explained in Appendix 13.1.

## 10.1 Phase Compensation in eSFQ

Phase compensation in superconducting electronics has to do with the balancing of phases at the different biasing nodes. Note that the phase difference in a single gate does not need compensation, instead it is exactly this phase difference that is the source of a flux pulse traveling from one point to the other. In a logic gate, synchronous or asynchronous, each junction normally switches only once after each input pulse or clock pulse. Therefore the phase imbalancing inside a logic circuit compensates itself.

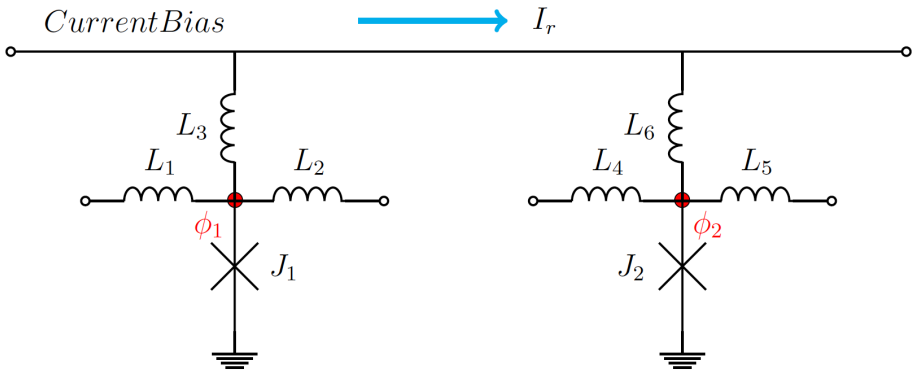



Figure 10.1: Phase difference in the biasing network

In RSFQ the phase difference between the different biasing nodes of parallel connected gates do not play a significant role. Since the biasing resistor collapses the current wavefunction which breaks superconductivity. As explained in the previous section the existence of a current going back into the biasing scheme can still exist, but only because of a difference in voltages between the different biasing nodes. We define the current $I_r$ as the current going back into

the biasing network. If the resistor falls away, the current $I_r$ will exist because of the difference in phase between the biasing nodes [51]:

$$I_r = \frac{(\phi_1 - \phi_2)}{2L} \tag{10.1.1}$$

where $\phi_1$ and $\phi_2$ are the phase values between two different bias nodes, as shown in Fig. 10.1. If two gates are parallel biased and the junctions of $Gate1$ switches many times, while the junctions in the $Gate2$ stays undisturbed, then a current may start to flow from $Gate1$ to $Gate2$ throught the biasing network. As an example, from Fig. 10.1 if the left JTL transmits flux pulses and the right JTL does not, then after

$$M = \frac{2L(I_c - I)}{\Phi_0} \tag{10.1.2}$$

pulses the current $I_r$ (light blue) will be large enough to switch the junction in the right JTL.

One approach to solve this problem called ERSFQ [52] was proposed, but has a large layout because of the feeding JTL and uses a large biasing inductor to limit the biasing current. eSFQ handles phase compensation by clocking every circuit. In the clocking scheme there will always be one of two junctions that must switch to keep the phase gradients between different biasing terminals constant [48]

$$\frac{d\phi_i(t)}{dt} \approx \frac{d\phi_j(t)}{dt} \tag{10.1.3}$$

The clocking network is connected to the decision-making-pair of the logic gate. By creating a system where either one junction or the other switches at every clock pulse, the absolute phase difference can be kept to no more than $|2\pi|$, since each biasing terminal gets incremented by $2\pi$ per clock cycle. In RSFQ circuits the biasing terminal may or may not be incremented, depending on the input the system receives. Which in turn may cause some biasing terminals to increment to much, causing to large a phase difference between different biasing terminals [53]. In ERSFQ to design for dynamic current the biasing inductor is made very big. This limits the current deviation by [52]

$$\delta I < \Phi_0 / L_b \tag{10.1.4}$$

In eSFQ this is handled by placing a junction in series with the biasing inductor. This junction never switches and stabilizes the current.

# Chapter 11

# eSFQ Cells

The current research in superconducting electronics is focused on decreasing the power dissipation of logic gates. The two most researched types of low energy SFQ circuits are AQFP [54] and eSFQ [52], [55], where AQFP lowers both dynamic- and static power dissipation, and eSFQ only focuses on static power dissipation. The advantage of eSFQ is its high clock speed over that of the AC biased AQFP technology.

In previous work only some of the RSFQ logic gates have been converted to eSFQ as discussed by Mukhanov and Volkmann [48], [56]. This chapter discusses the attempts made to create the first AND, Merger and NOT gates for eSFQ logic using the AIST-HSTP fabrication process. The eJTL and eDFF [55] was also fabricated using this process. Investigating eSFQ technology using different fabrication processes is an important part in proving the stability of eSFQ. Most of the already designed eSFQ cells was fabricated with Hypres's 4.5 $kA/cm^2$ process.

Clocking of SFQ electronics is normally done in the opposite direction of data flow. This method is based on a shift register type system, where the output is clocked out first and then the new data is read into the system. Advantages of this method includes clocking at higher frequencies and increasing circuit margins.

In simulation the designed circuits worked as should with acceptable margins as will be shown later. The circuit layouts introduced parasitic inductances which changed the functionality of the system, and led to some changes being made in the final design and layout. One such an example is by changing junction rotations in layout, to minimize parasitic inductances. There are four basic concepts that must be kept in mind when designing eSFQ circuits:

- A SFQ pulse is propagated only when clocking, which means an eSFQ circuit will always be clocked. There is no asynchronous circuits in eSFQ

and therefore, there must always be a decision-making-pair in the circuit.

- There must always be phase compensating junctions. In other words, if one junction switches because of a received clock pulse, it's DMP partner junction must switch when no clock pulse is received.
- No junction may switch more than once during one clock period.
- The compensating junctions will in most cases have the lowest margins.

Each time a junction switches a voltage pulse is created for a short time. The area under this pulse has a value of 2.07 $mV.ps$. It may be difficult sometimes to see whether a junction has switched or not by just analyzing the SFQ pulses. Instead, evaluating whether the junction made a $2\pi$ phase jump or not is a more practical way of data analyses.

## 11.1 Programs

Designing any electronic circuit requires several design steps. Programs used to develop the circuits are all free to use and are listed below

- JSIM - Simulation of circuits (Superconducting Spice)
- US-Tools - Set of margin, yield and verification analyses tools developed at University Stellenbosch
- Lasi - Layout of circuits for fabrication
- Inductex - Inductance parameter extraction tool, also developed at University Stellenbosch
- Graph - A self-written script in Matplotlib (Python) to plot simulation results.

The inductance extraction is done by creating a *.cir* file which is the circuit netlist of the physical layout. In semiconducting technology this netlist is generated by a schematic-vs-layout tool. There is no such tool avaliable for superconductive circuit layouts, with the exception of custom scripts for the discontinued commercial software Cadence DIVA. Therefore, this netlist has to be manually created by the designer. After circuit layout and inductance parameter extraction, the new circuit with its parasitic inductance values needs to be re-simulated in JSIM and the layout changed to correct inductance values where necessary. All circuits discussed in this chapter was designed for the AIST-HSTP process, except the eNOT gate which was designed but not fabricated. Circuit tests were not successful, but it was found after fabrication that a misinterpretation of junction areas during layout rendered all junction critical currents wrong. After parameter extraction and using the correct JJ critical current values all circuits worked as explained in this chapter.

## 11.2 eDFF

The eDFF is the most basic eSFQ circuit, since it is the simplest clocked logic gate. The eDFF is a memory cell which stores an input pulse in a decision-making-pair to produce an output at the next clock cycle. The schematic diagram is shown in Fig. 11.1, where the junction models are simplified. An

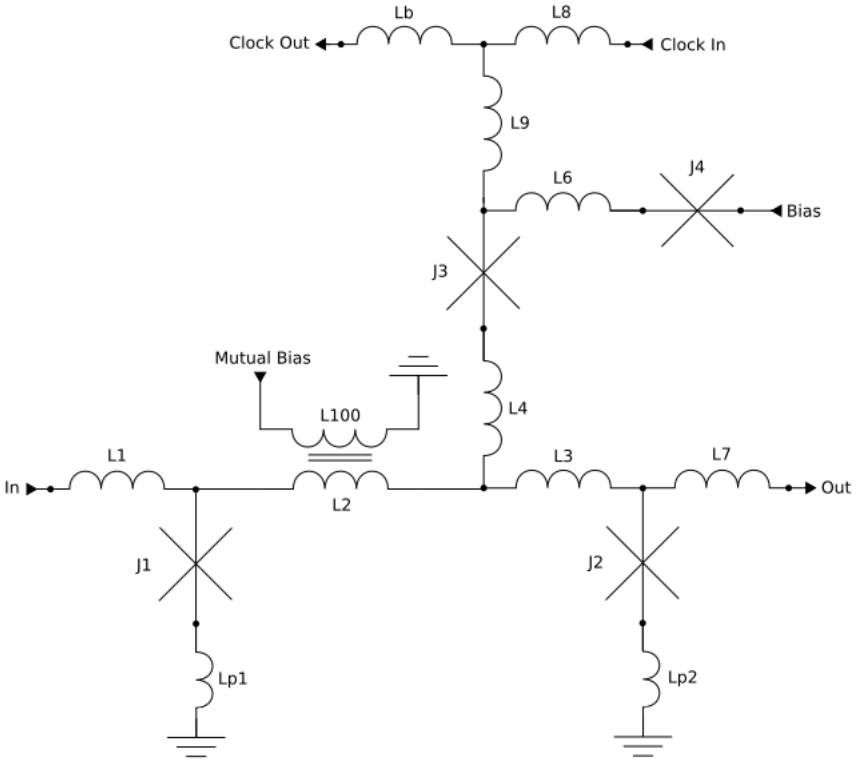



Figure 11.1: eDFF Schematic

incoming pulse will switch $J_1$ and store the pulse in the loop $J_1 - L_2 - L_3 - J_2$. With $J_2$ biased by the biasing current and the trapped pulse biasing it even more, a receiving clock pulse will let $J_2$ switch and produce an output. The mutual biasing between $L_2$ and $L_{100}$ biases $J_1$ and also increases the margins of this junction. The biasing of $J_1$ can be seen in the JSIM phase results shown in Fig. 11.2 by looking at the the initial phase offset of $J_1$ before it switches for the first time. Junction $J_4$ never switches as observed in Fig. 11.2, it only stabilizes the biasing network.

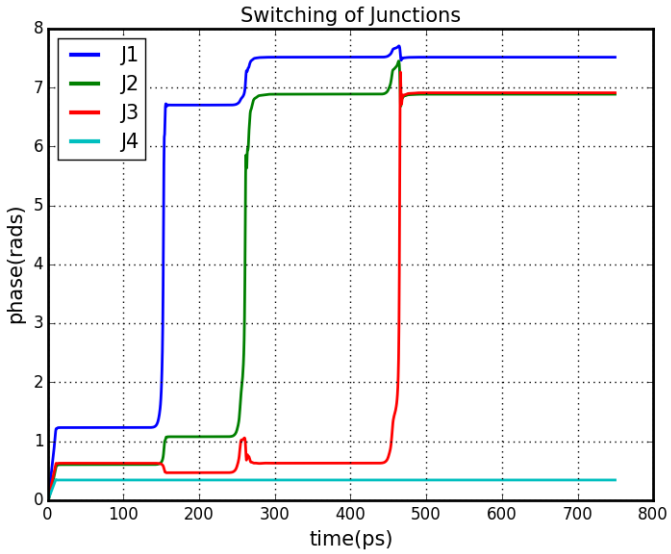



Figure 11.2: eDFF JSIM phase simulation results

## 11.3 eJTL

The original JTL gate only latches an input pulse through to the output without saving any data, and without being clocked. In eSFQ all asynchronous circuits has to be made synchronous and therefore a clocking network has to be integrated into the eJTL circuit. As proposed in [55] the eJTL gate works like a shift register, by latching the stored input pulse to the output.

The receiving pulse switches $J_1$ and stores the pulse in the $J_1 - L_3 - L_4 - J_2$ loop. A receiving clock pulse will switch $J_2$ and the switching of $J_2$ will cause $J_3$ to switch, and generate an output through $L_7$. All three of the bottom junctions made a $2\pi$ phase slip, therefore the phase $\phi_1$ at the biasing node will also be incremented by $2\pi$ after a clock pulse. However, if there is no pulse saved in the $J_1 - L_3 - L_4 - J_2$ loop then all three of the top junctions will switch. Again resulting in the phase $\phi_1$ increasing by $2\pi$. If $J_2$ switches some current will go upwards through the $J_{22} - L_{10} - L_9 - L_8 - J_{next\_clock}$ loop, which will bias junction $J_{next\_clock}$ by a small amount.

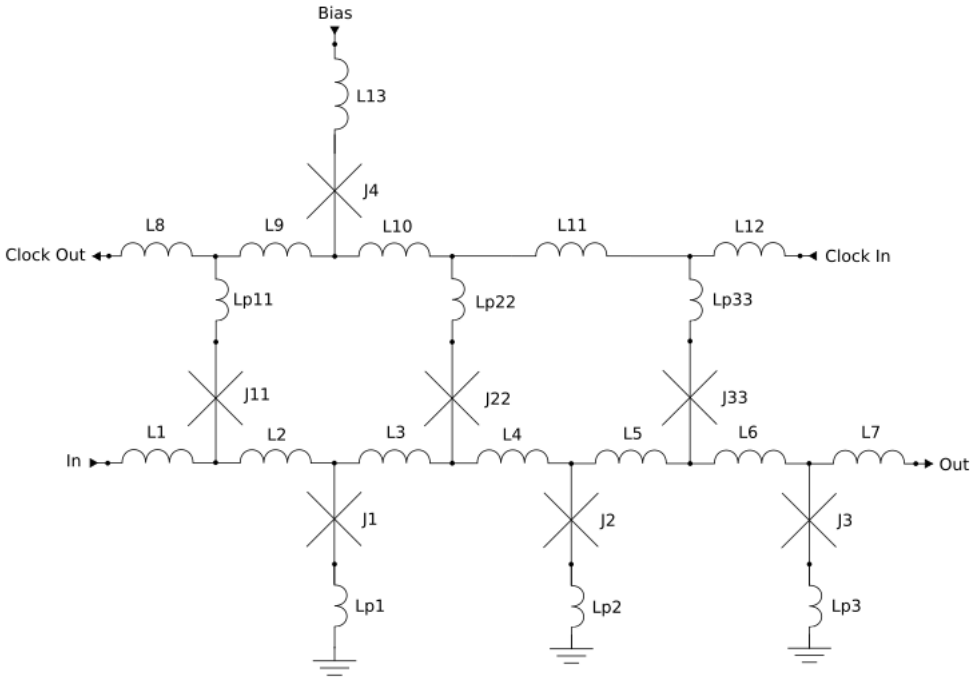



Figure 11.3: eJTL Schematic

Junction $J_{next\_clock}$ is the first junction connected to the **Clock Out** port. When $J_3$ switches the same happens but this time the resulting current is enough to cause $J_{next\_clock}$ to switch and let the clock pulse propagate for further processing. The circuit netlist representing the layout for the eJTL is shown in Fig. 11.3. Each junction has a parasitic inductance in series, with an approximate value of 0.3 $pH$. If two junctions are put on top of eachother, the parasitic inductance $L_{J11}$, $L_{J22}$ and $L_{J33}$ values goes up to about 1.5 $pH$. This is because the connection between the upper and lower junctions, $L_{J11}$ and $L_{J1}$, increases the inductance between them. Re-simulating the eJTL with these parasitic values shows that current are stored in the upper loops of the circuit. This is because the total inductance value of the loop, $L_{J11} - L_9 - L_{10} - L_{J22} - L_2 - L_3$ is large enough to store a flux pulse.

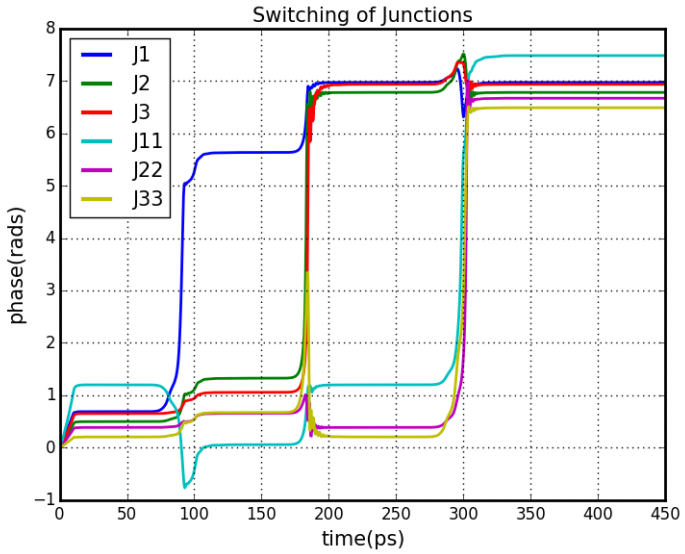



Figure 11.4: eJTL JSIM phase simulation results

To account for this behaviour the total loop inductance needs to be decreased. Changing the layout of the circuit is the only way to accomplish this, and to decrease the loop inductance the junctions are placed next to each other.

## 11.4 eMerger

The merger gate combines two input pulse to produce only one output. It is the reverse of the splitter gate that spits one input pulse into two output pulses. If only one output is received in the merger circuit, then a logic one is still produced. This eMerger circuit forms the bases of the eAND gate, which will be discussed later in this chapter.

Two input pulses will each switch $J_{m1}$ and $J_{m2}$. The two pulse will combine at the node before $L_1$ and propogate through $L_1$ to switch $J_1$. The pulse is then stored in the loop $J_1 - L_2 - L_3 - J_2$. A clock pulse is inserted and the rest of the circuit has the same functioning as the eDFF. If only one input pulse is received, either at $In1$ or $In2$, then the corresponding junction will switch, which in turn will cause the other input junction to switch. The resultant pulse will propogate through $L_1$ to switch $J_1$. The reason why the switching of one junction, either $J_{m1}$ or $J_{m2}$ can cause the other to switch is because of the mutual biasing current connected to $L_1$. This current divides equally between the two branches containing $J_{m1}$ and $J_{m2}$ and biases these two junctions. The sum of the inductors $L_{m3}$ and $L_{m4}$ are smaller than that of $L_1$, which results in most of the current produced by the switching of the input junction $J_{m1}$ to

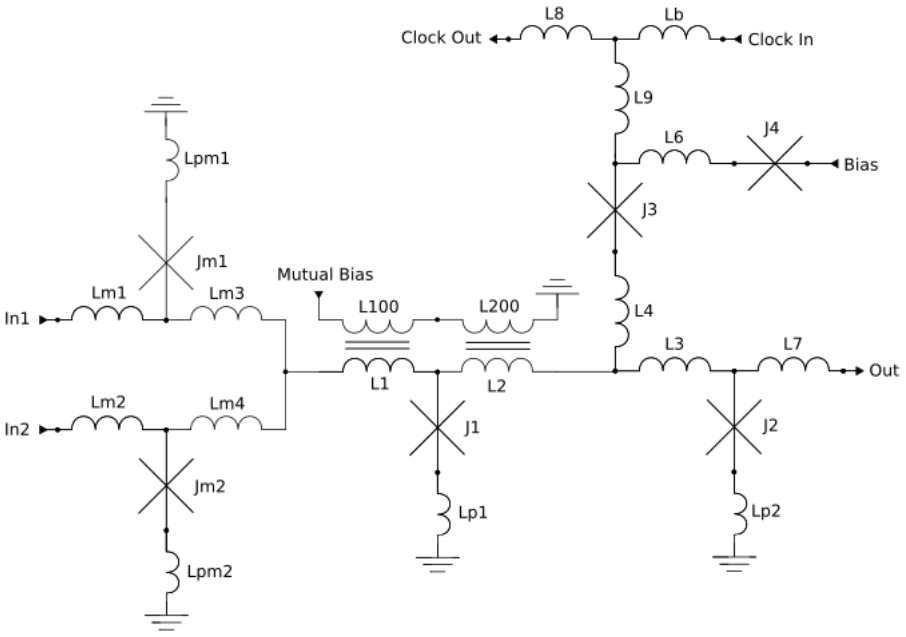



Figure 11.5: eMerger Schematic

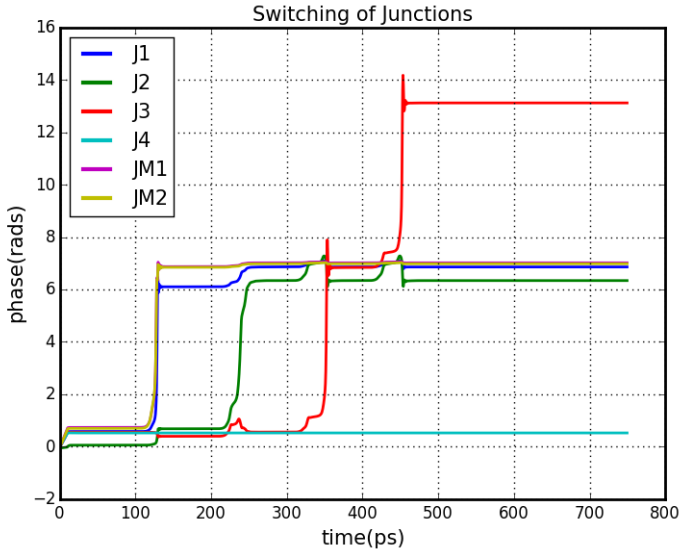



Figure 11.6: eMerger Jsim two input pulses

propogate through the other input junction $J_{m2}$ which did not switch. This as explained will cause $J_{m2}$ to switch and since $J_{m1}$ has already switch the resultant pulse will cause $J_1$ to switch. The margins of this circuit is relatively stable, except for the phase compensating junction, $J_3$, which has margins of $-19.7\%, +12.0\%$. The JSIM simulation results are shown in Fig. 11.6 and

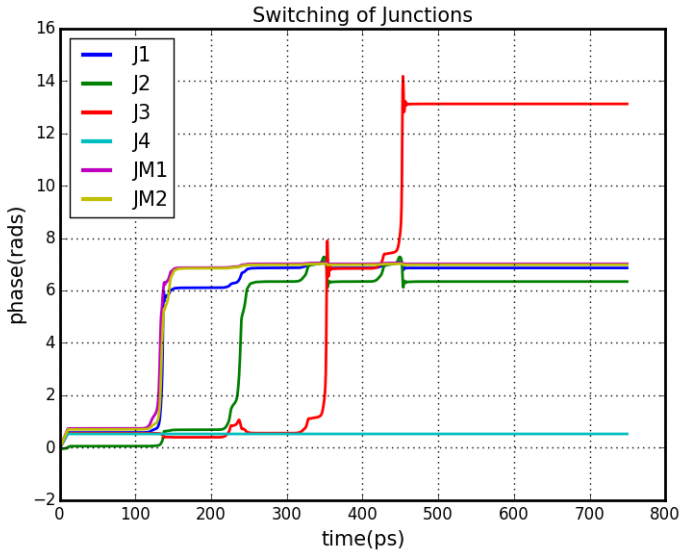



Figure 11.7: eMerger Jsim input pulse at In1

Fig. 11.7 where three clock pulse were sent in at 200 $ps$, 300 $ps$ and 400 $ps$. Fig. 11.6 shown the output if both inputs are excited with a pulse and Fig. 11.7 shows the result of only one input excited.

## 11.5 eAND

An AND gate operates by giving a high output if two input pulses are received. The eAND gate's schematic is very similar to that of the eMerger, but only with one inductor biased as shown in Fig. 11.8. The junction $J_1$ will only switch if both junctions $J_{m1}$ and $J_{m2}$ switches. If both junctions switches then the resultant pulse will combine at the node before $L_1$ and propogate forward to switch $J_1$. If only one junctions switches, lets say $J_{m1}$, then the resultant current will not be able to switch $J_{m2}$, because it is not biased as in the eMerger circuit. The result is that $J_1$ will not switch and no pulse will be saved in the decision-making-pair, $J_1 - L_2 - L_3 - J_2$. The input junctions has the lowest margins in the eAND circuit, unlike the eMerger where the lowest margins is the phase stabilizing junction. The reason for this is believed to be the fact that these input junctions are not biased as in the eMerger circuit. This requires the junctions to have a lower critical current. The logic of the eAND gate is more dependent on the correct switching of the input junctions than in the eMerger, which can also be the cause for lower margins.

One problem that might arise from this circuit is the synchronization of the two input pulses, to produce a logic one. Simulations where done were the two different input pulses was separated by a $5ps$ time interval, and because of the

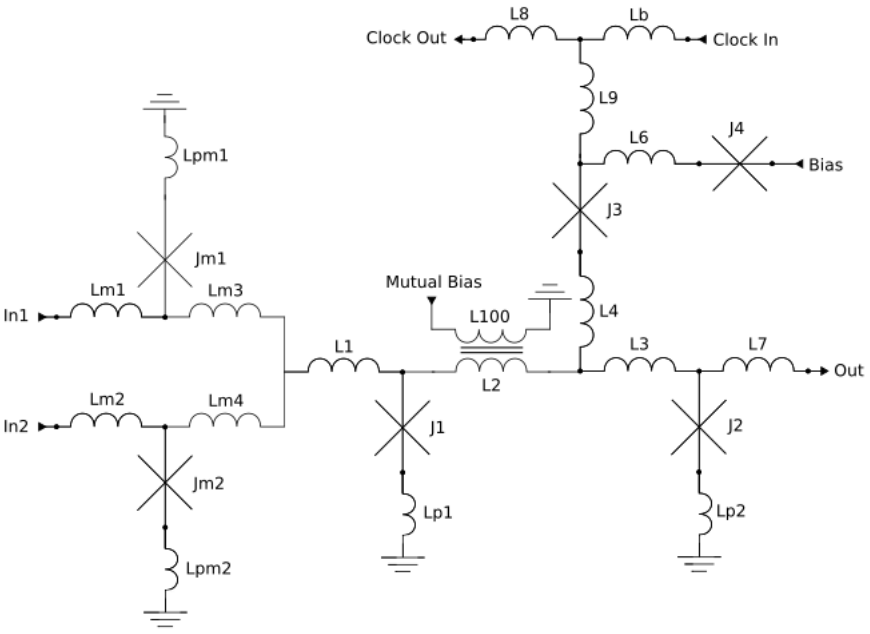



Figure 11.8: eAND Schematic

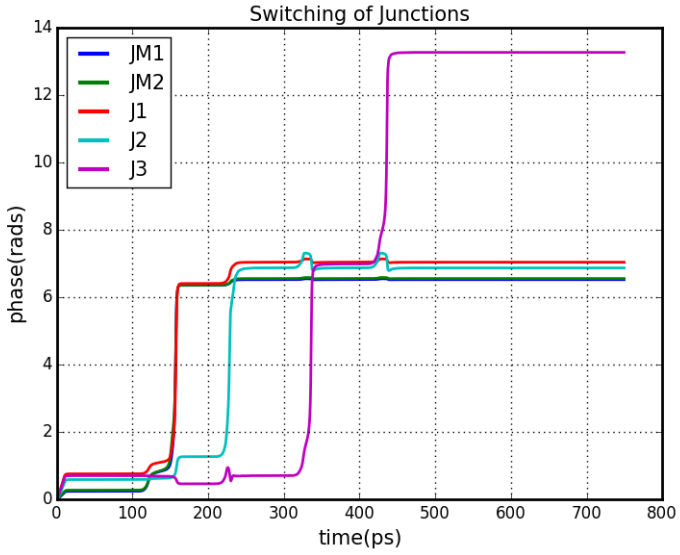



Figure 11.9: eAND JSIM Results with both inputs receiving an input pulse

pulse rise and fall time, the gate would still produce a logic one. Therefore, changing the damping resistor values of the input junctions to increase the pulse rise time, will increase the separation time between the two input pulses that will still produce a logic one output.

## 11.6 eNOT

The basic eNOT gate is a logic circuit that inverts the input. It is one of the most important gates in any technology, since it builds up to the NAND gate. If a pulse is received at the input, then no output should be produced on the next clock pulse. When no pulse is stored in the decision-making-pair and a clock pulse is received, then the circuit should produce a pulse at the output.

A conceptual design for the eNOT gate is presented here, although it was not fabricated due to layout constraints. As a result the re-simulated eNOT

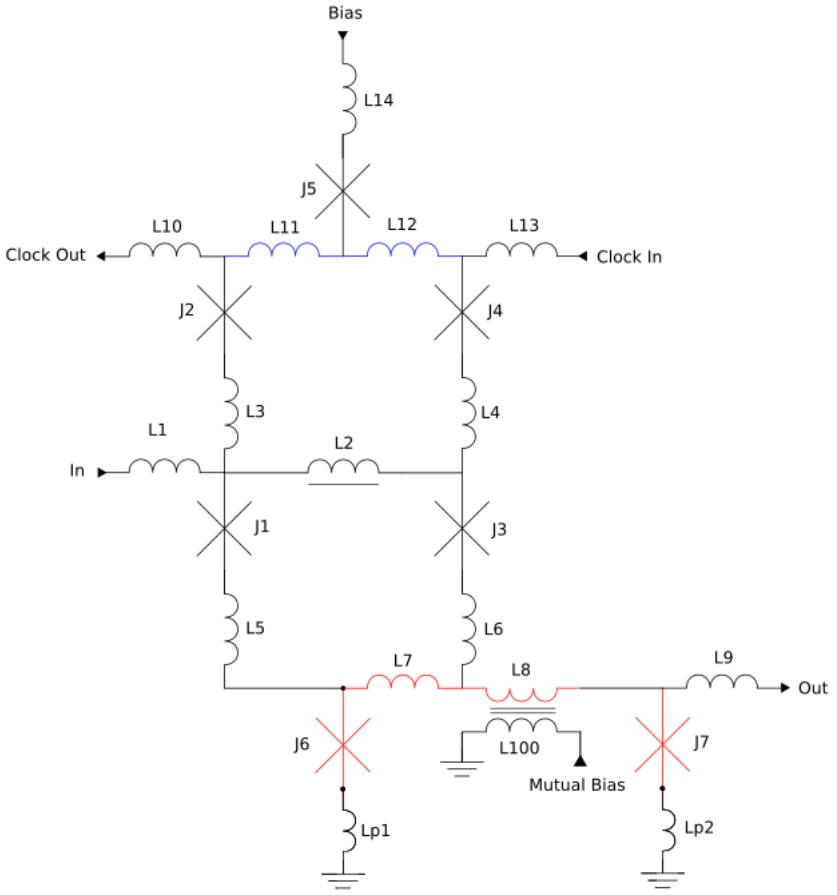



Figure 11.10: eNOT Schematic

using the values in the actual layout did not function as needed. The input pulse will be saved in the loop $J_1 - L_2 - J_3 - L_6 - L_7 - L_5$ and if the circuit is clocked, then junction $J_3$ will switch. This will cancel the stored fluxon saved in the loop which result in no output signal.

Biasing $J_1$ and $J_3$ are done by inserting current at the top *Bias* node. The current splits into the two different loops and joins again at the node above $J_6$. This causes junction $J_6$ to become over biased, and therefore the critical current needed for this junction is atleast $800\mu A$. Biasing $J_6$ with a second source in the opposite direction is the only practical way of solving this problem without drastically changing the circuit design. The biasing is done mutually through inductors $L_8$ and $L_{100}$, which decreases the total bias current through $J_6$ and positively biases $J_7$. If no pulse is stored in the decision-making-pair then the next clock input will cause $J_6$ to switch, which in turn will switch $J_7$ and produce an output. An input pulse switches $J_1$ and stores the pulse as explained. This causes a $2\pi$ phase increase above $J_2$. An input clock pulse will switch $J_3$ and increase the phase at the node above $J_4$. As a result this increases the phase of the biasing node. If no input pulse was received then $J_6$ will switch at the next clock pulse and compensate for the biasing phase.

One of the reasons for circuit failure after the physical layout is due to the inductors $L_3$ and $L_4$ being too large. This causes unwanted current to be stored in the loop $J_2 - L_{11} - L_{12} - J_4 - L_4 - L_2 - L_3$. Looking at the layout in Fig. 11.15 the reason for these large inductors is due to the large inductor $L_2$. Solving this problem is possible by using a different layer for $L_2$ so it can go beneath inductors $L_{11}$ and $L_{12}$, shown in blue in Fig. 11.10.

## 11.7 Circuit Layouts

In this section the component margins and circuit layouts of the eDFF, eJTL, eMerger, eAND and eNOT gates are shown. Existing conventional RSFQ cells from the CONNECT cells library were used to connect the eSFQ cells to the necessary interfaces. The circuit layouts were done for the AIST-HSTP process, where each input/output was connected to two JTLs. These JTLs was then connected to the DC/SFQ and SFQ/DC interface circuits. In the AIST process the CTL layer are used to connect with the interfacing cells. To decrease the probability of failure the input/output ports were placed as far apart as possible, to overcome any mutual coupling effects between the input/output strip line. Fig. 11.11 shows the layout of the eDFF gate, with

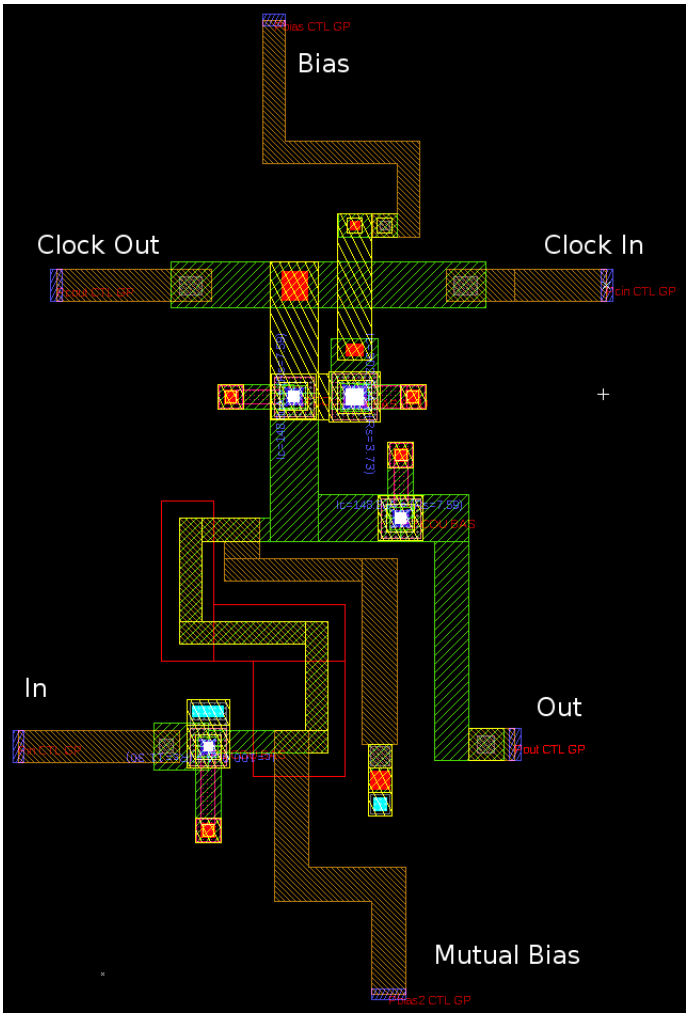



Figure 11.11: eDFF Layout

the mutually coupled bias line using a ground plane hole. The coupling factor obtained in this layout was $k = 0.3$.

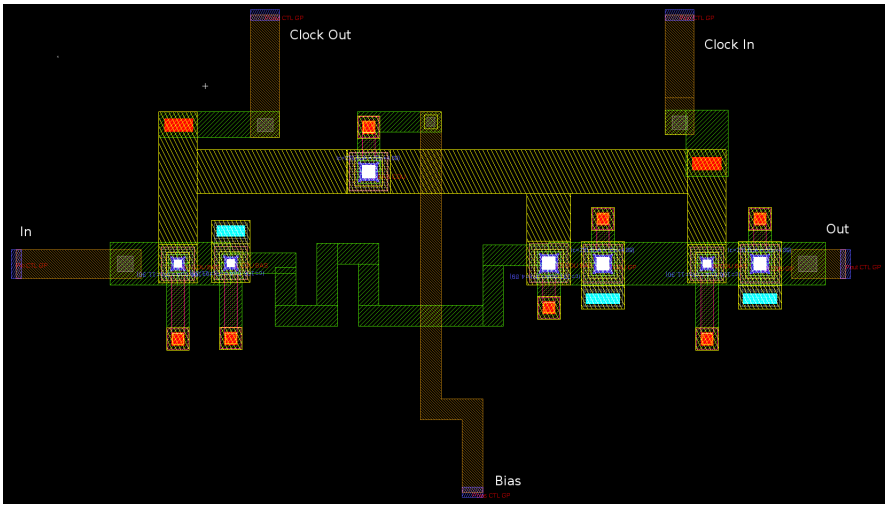



Figure 11.12: eJTL Layout

The eJTL gate are presented in Fig. 11.12 with the junctions, $J_1 - J_{11}$, $J_2 - J_{22}$ and $J_2 - J_{22}$ in Fig. 11.3 placed as close as possible to each other to decrease the coupling inductance between them. The bias line are taken downwards through the center of the circuit as shown in Fig. 11.12, to ensure no coupling between the bias line strip and the clock in and clock out strip lines.

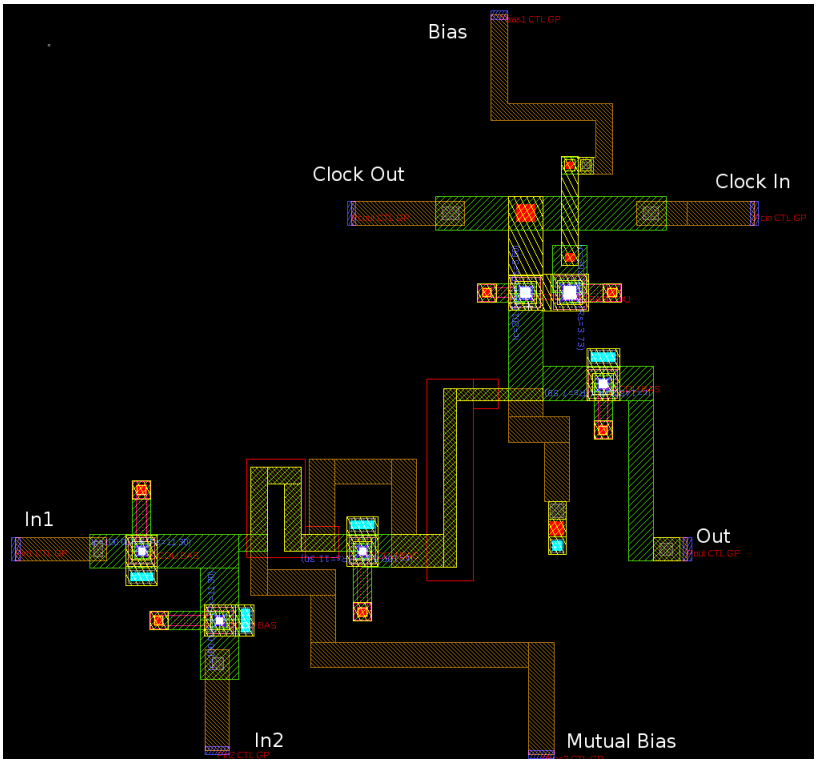



Figure 11.13: eMerger Layout

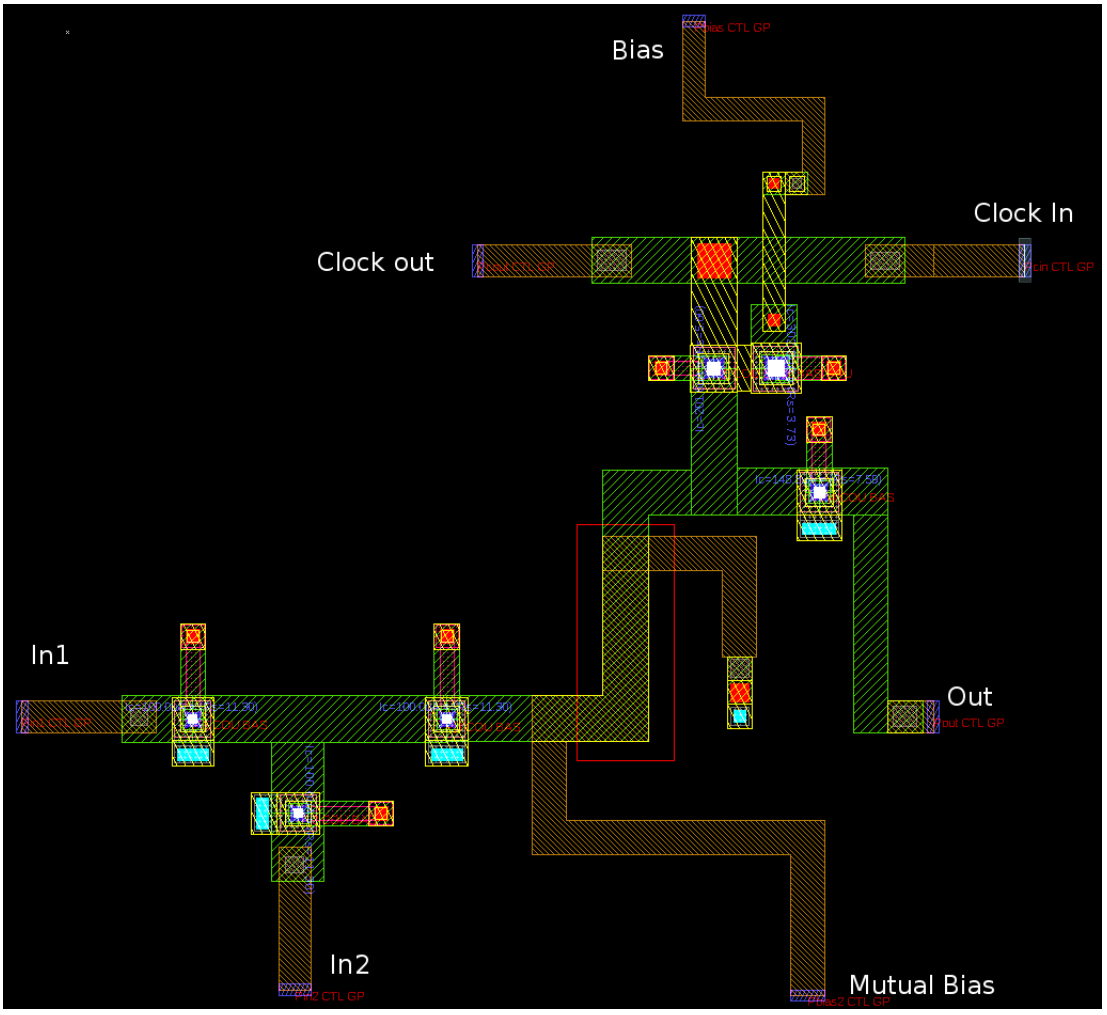



Figure 11.14: eAND Layout

The eMerger and eAND gate circuit layouts shown in Fig. 11.13 and Fig. 11.14 are very similar to that of the eDFF gate, but with different inductance values and two input ports connected.

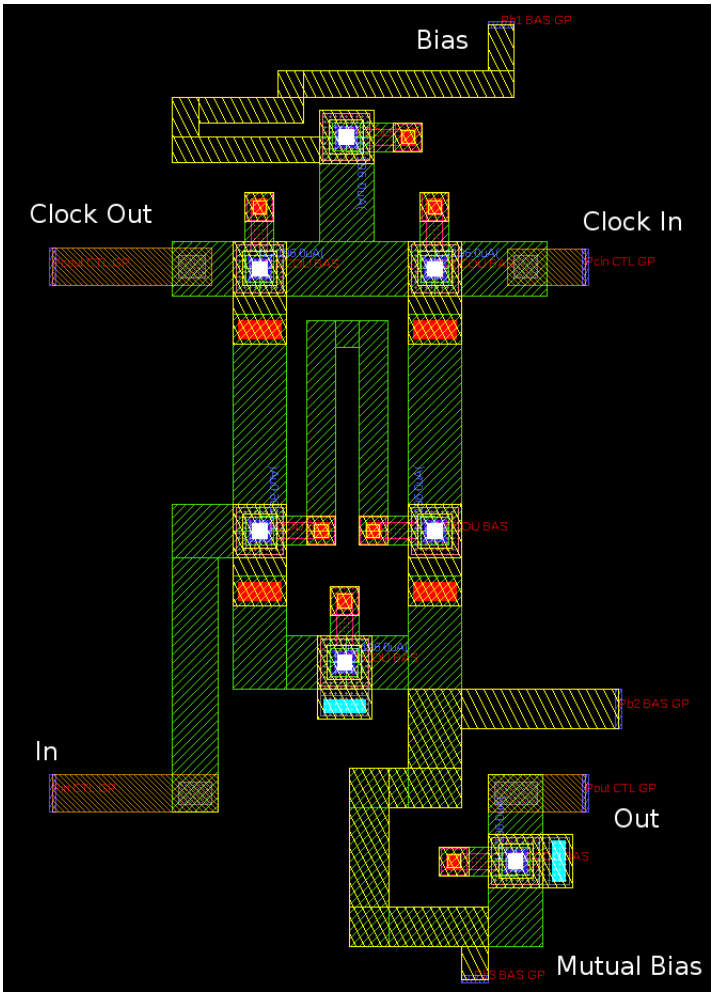



Figure 11.15: eNOT Layout

The reason for the eNOT gate not working after simulating the extracted parameter values, was due to the large inductance values in the center of the circuit layout, Fig. 11.15. A possible solution to this problem is to use different fabrication layers for the inductances $L_3$ and $L_4$, which is in the middle part of the layout.

## 11.8 Margin Results

In this chapter the simulation margins obtained for the eSFQ circuits fabricated are shown and discussed. The goal was to obtain margins with a 30 % spread to lower the probability of circuit failure.

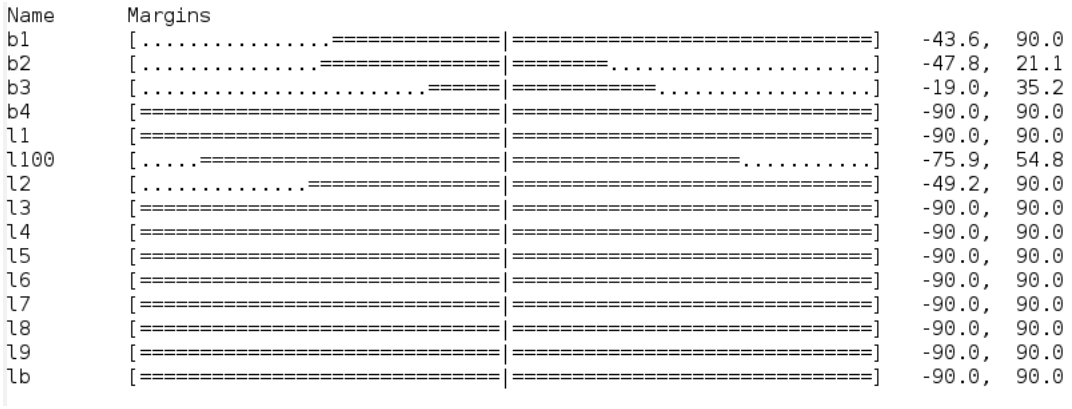



Figure 11.16: eDFF Margins

In the eDFF gate the phase compensating junction $J_3$ has the lowest margins, Fig. . The margins of $J_1$ was increased by adding the mutually connected bias current to the DMP, as shown in the schematic (Fig. 11.1).

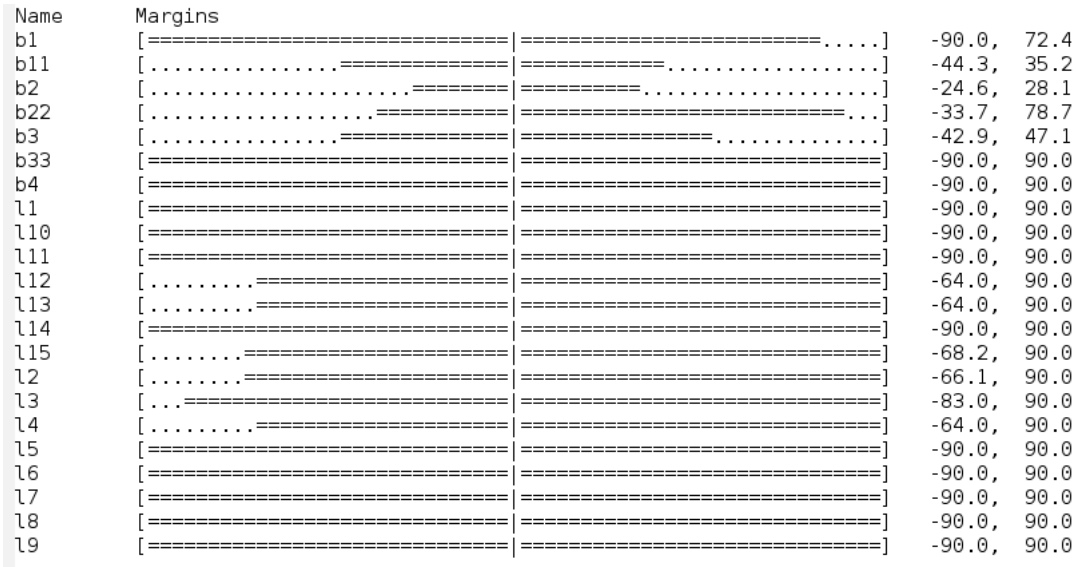



Figure 11.17: eJTL Margins

Fig. 11.17 shows the margins of the eJTL circuit, here the junction $J_2$ has the lowest margins and not the phase compensating junctions as in the case of the eDFF. The reason for this is believed to be because the biasing current are

split between three phase compensating junctions, $J_{11}$, $J_{33}$ and $J_{33}$. Also, the inductor which is in series with these junctions are smaller than in the eDFF circuit.

```
Name      Margins
b1        [============================|======================......]   -90.0,  71.7
b2        [...=========================|==================..........]   -82.3,  54.1
b3        [........................======|====........................]   -19.7,  12.0
b4        [============================|============================]   -90.0,  90.0
bm1       [============================|=========================...]   -90.0,  78.0
bm2       [============================|============================]   -90.0,  90.0
l1        [============================|============================]   -90.0,  90.0
l100      [............================|==================..........]   -52.0,  51.3
l2        [..............==============|=========================....]   -48.5,  75.2
l200      [...=========================|============================]   -83.7,  90.0
l3        [============================|============================]   -90.0,  90.0
l4        [................============|======================......]   -40.8,  67.5
l6        [============================|============================]   -90.0,  90.0
l7        [============================|============================]   -90.0,  90.0
l8        [............================|============================]   -54.8,  90.0
l9        [============================|============================]   -90.0,  90.0
lb        [============================|=============...............]   -90.0,  39.4
lm1       [============================|============================]   -90.0,  90.0
lm2       [============================|============================]   -90.0,  90.0
lm3       [============================|============================]   -90.0,  90.0
lm4       [============================|============================]   -90.0,  90.0
lm5       [============================|============================]   -90.0,  90.0
lm6       [============================|============================]   -90.0,  90.0
```

Figure 11.18: eMerger Margins

The margin analyses is very much the same for the eMerger and eAND gate, as for the eDFF gate. With the lowest margins being that of the phase compensating junctions, Fig. 11.18 and Fig. 11.19.

```
Name      Margins
b1        [============================|========.....................]   -90.0,  26.0
b2        [.................==========|=============.................]   -34.5,  40.8
b3        [.................==========|==============................]   -37.3,  42.9
b4        [============================|============================]   -90.0,  90.0
bm1       [============================|========.....................]   -90.0,  26.7
bm2       [============================|======.......................]   -90.0,  16.2
l1        [============================|============================]   -90.0,  90.0
l100      [.................==========|============================]   -33.0,  90.0
l2        [.........==================|==================..........]   -61.2,  57.0
l3        [============================|============================]   -90.0,  90.0
l4        [============================|============================]   -90.0,  90.0
l6        [============================|============================]   -90.0,  90.0
l7        [..==========================|============================]   -85.8,  90.0
l8        [============================|============================]   -90.0,  90.0
l9        [============================|============================]   -90.0,  90.0
lb        [============================|============================]   -90.0,  90.0
lm1       [============================|============================]   -90.0,  90.0
lm2       [============================|============================]   -90.0,  90.0
lm3       [...=========================|============================]   -83.0,  90.0
lm4       [============================|============================]   -90.0,  90.0
```

Figure 11.19: eAND Margins

# Chapter 12

# Practical Applications: Magnetic Field Analyses in eSFQ Circuits

The proposed toolset can be used to analyze mutual coupling parts of a SFQ circuit, which can then help the designer understand how to improve or decrease wanted or unwanted mutual couplings in the circuit. In this chapter examples are shown that calculates the magnetic fields for the eDFF gate designed as explained in Chapter 11. Changes are made to the circuit and the magnetic field results are compaired. In each of the following figures that represent the GDS layout of the eDFF, the horizontal white line represents the slice taken for the cross-sectional view of the magnetic fields.

First we analyze the magnetic fields of the mutual coupling structure in the eDFF, with the ground plane hole in place. Then we analyze the effect the ground plane hole has on the structure's magnetic fields by removing the hole and adding the GP layer back into the simulation. Lastly, we analyze the mutual coupling effect of the biasing line with itself when incorrectly placed.

## 12.1 Designed eDFF mutual coupling magnetic fields

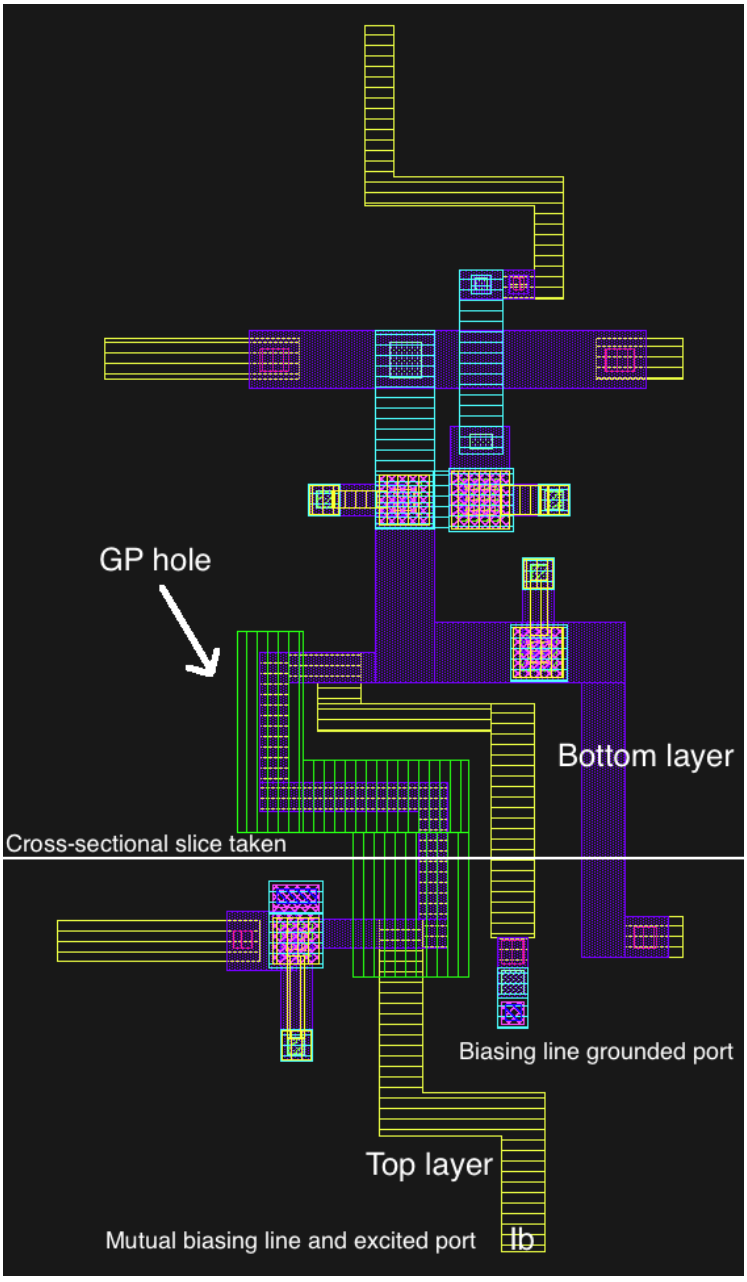



Figure 12.1: The final eDFF gate with a y-directed slice taken for magnetic field visualization. The mutual biasing line is excited, with the port at the bottom part of the figure

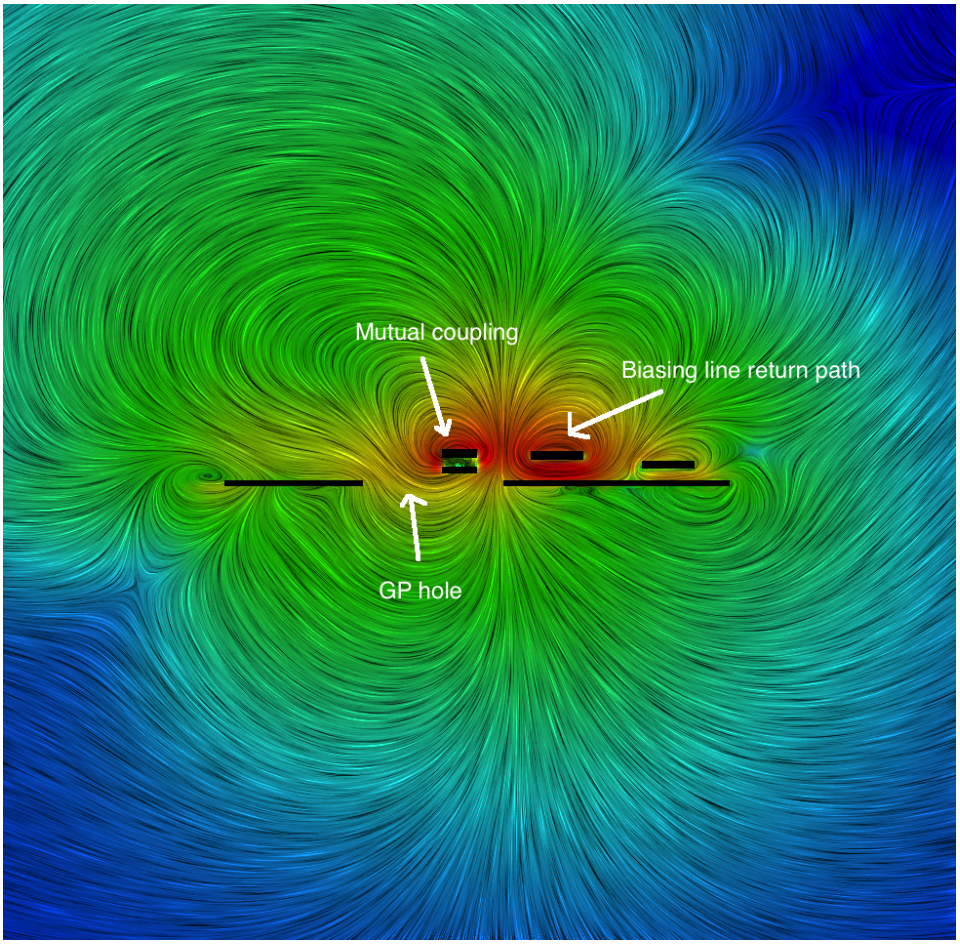



Figure 12.2: Magnetic fields as seen from the cross-section view, for Fig. 12.1

By cross-referencing Fig. 12.1 and Fig. 12.2 we can make sence of the different layers sliced and visualy analyze the magnetic fields around these superconducting layers. The measured coupling factor using InductEx is close to 4, and the coupling factor without the GP hole is 2.9. The mutual coupling increases with the addition of the GP hole, because the magnetic fields surrounding the mutual coupling geometry has a larger loop area, than without a ground plane hole. As seen in Fig. 12.2 the field lines has more space to bend around the mutual structure.

## 12.2 eDFF without GP hole

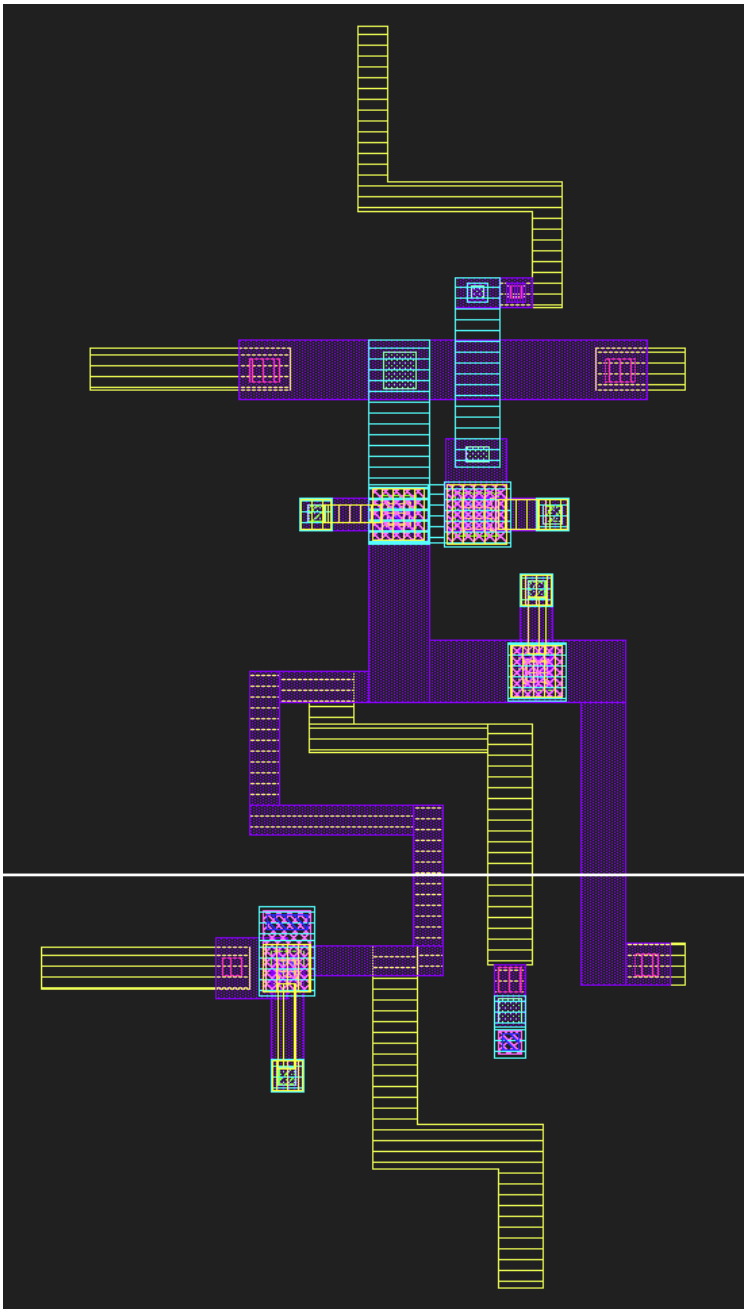


Figure 12.3: eDFF gate with the ground plane hole removed

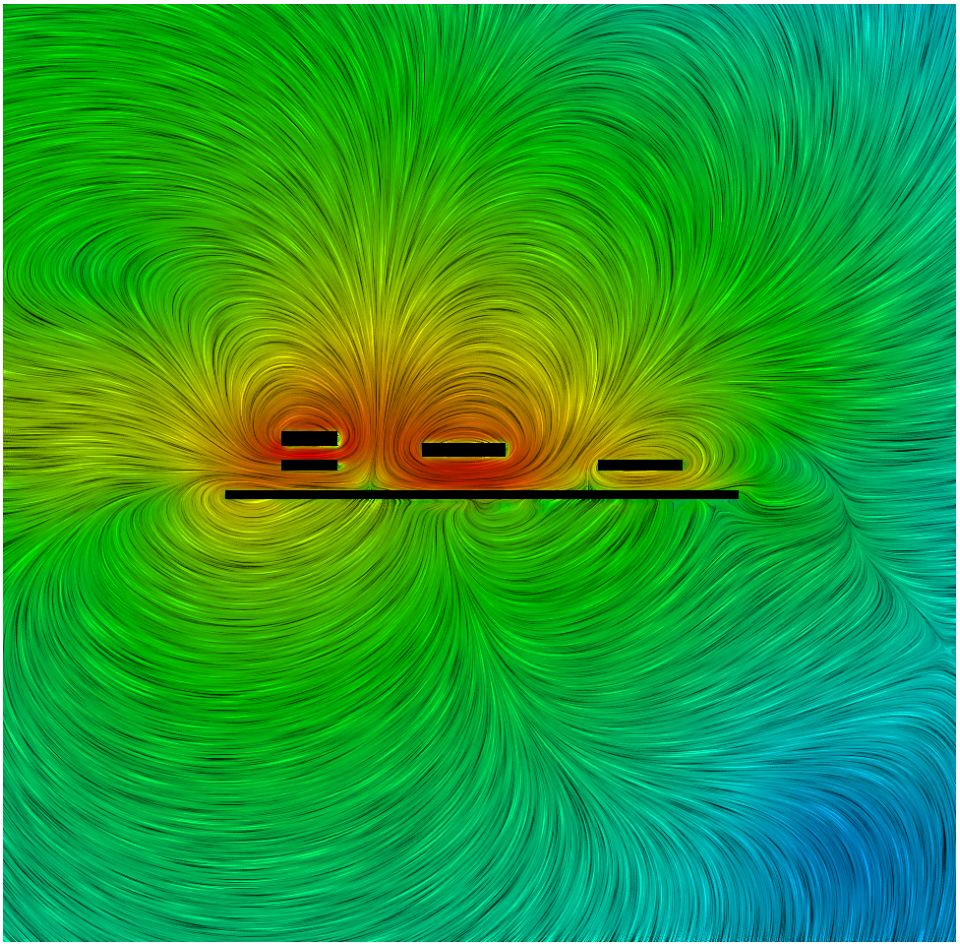


Figure 12.4: Magnetic fields as seen from the cross-section view, for Fig. 12.3

Compairing Fig. 12.4 with Fig. 12.2 we see that the magnetic field lines has less space to move around the mutual coupling, because of the GP layer blocking its path. We also note that there is a large coupling factor between the mutual coupling structure and the biasing line's return path to ground, which is the center polygon in Fig. 12.4.

## 12.3 eDFF without GP hole and biasing line spaced futher away

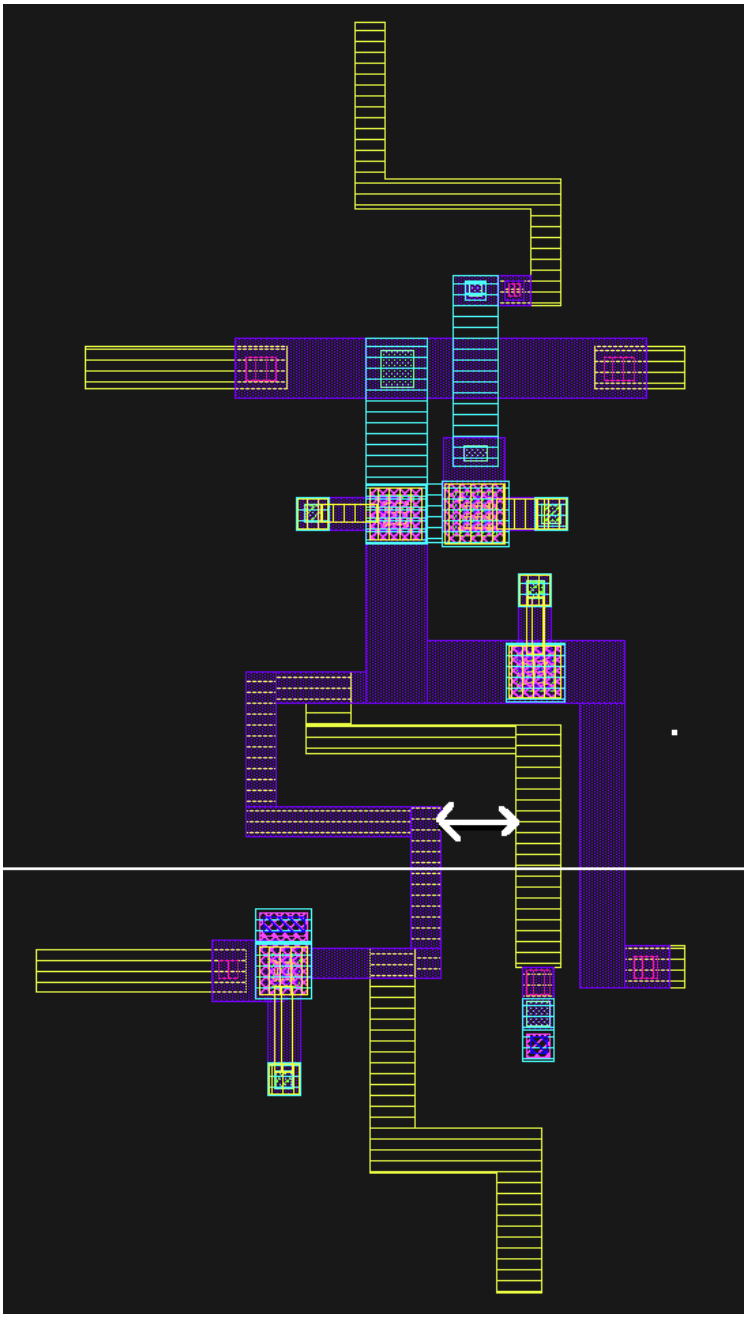


Figure 12.5: eDFF with ground plane hole removed and the biasing return line is placed futher away from the mutual coupling

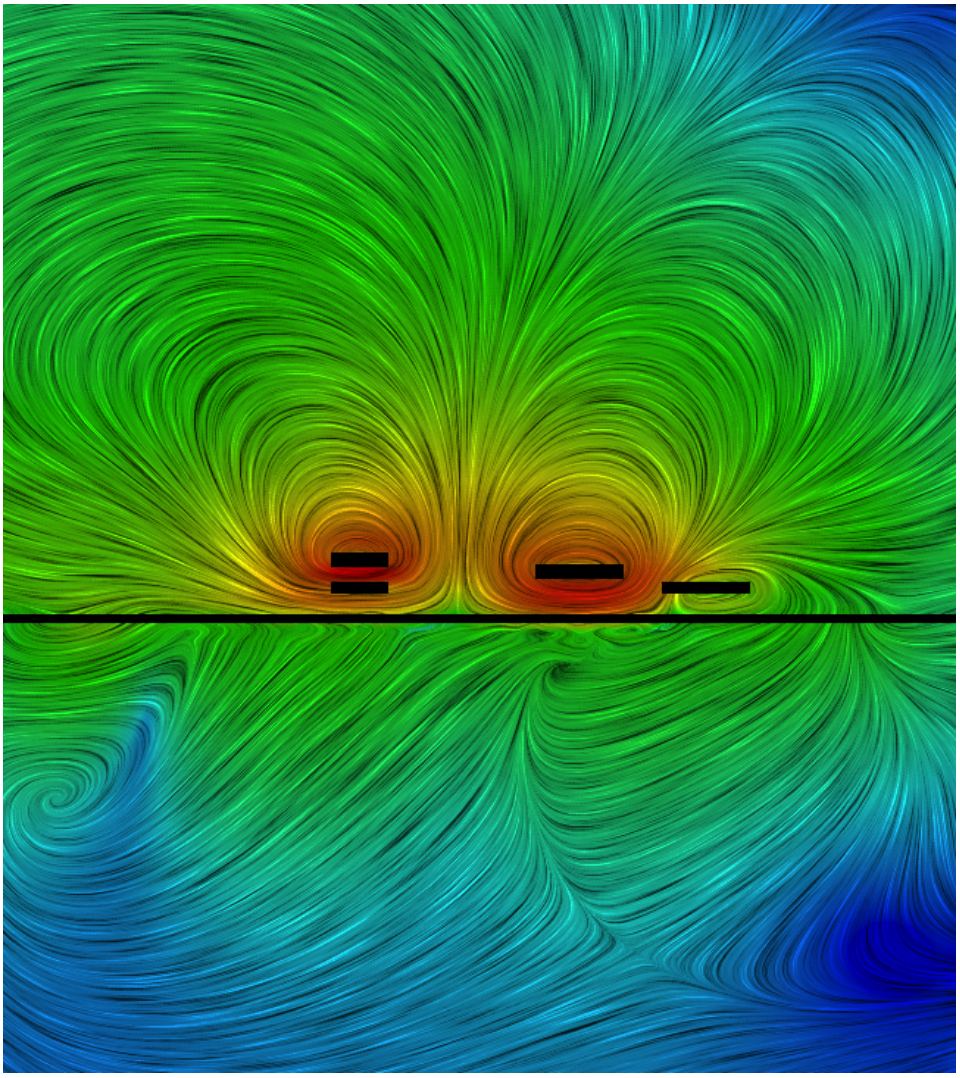


Figure 12.6: Magnetic fields as seen from the cross-section view, for 12.5

The biasing line's return path to ground and the mutual coupling structured are placed futher away from eachother, and the magnetic fields are analyzed. From Fig. 12.6 we see that the magnetic field lines are much more detached than in Fig. 12.4, which means the coupling factor between these two lines are less. In this example the ground plane overhang parameter in the .ldf file was increased for InductEx.

## 12.4 eDFF with biasing line grounded perpendicular to initial biasing line

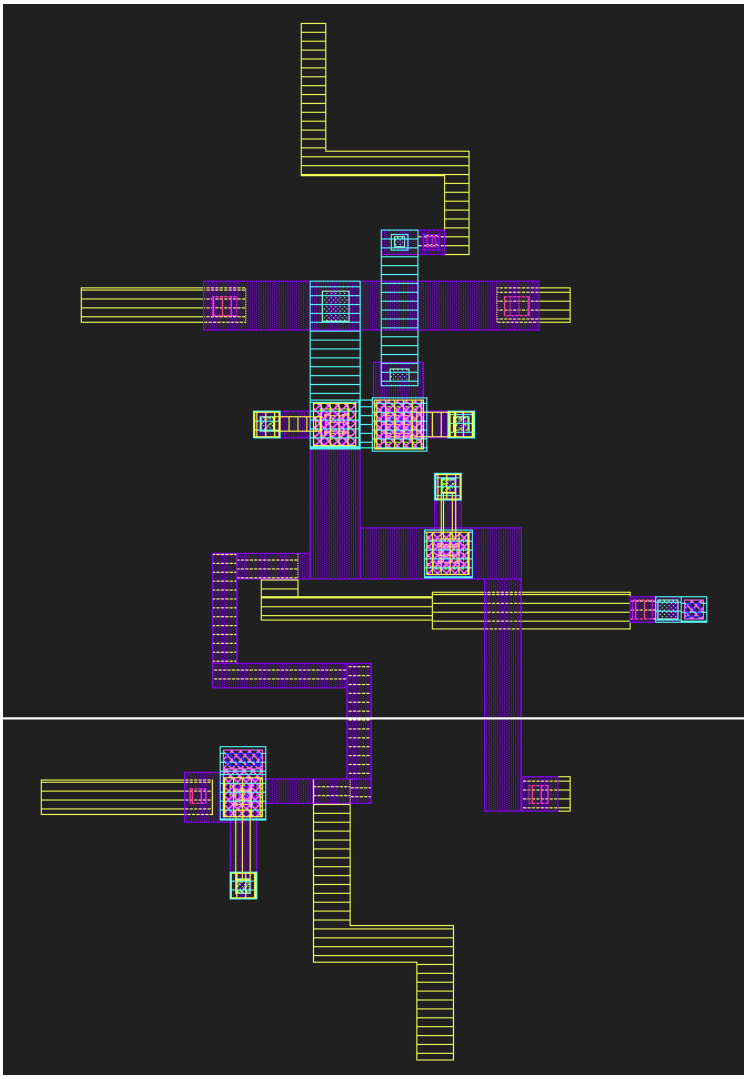


Figure 12.7: eDFF without ground plane hole and biasing return line is places perpendicular to the mutual coupling

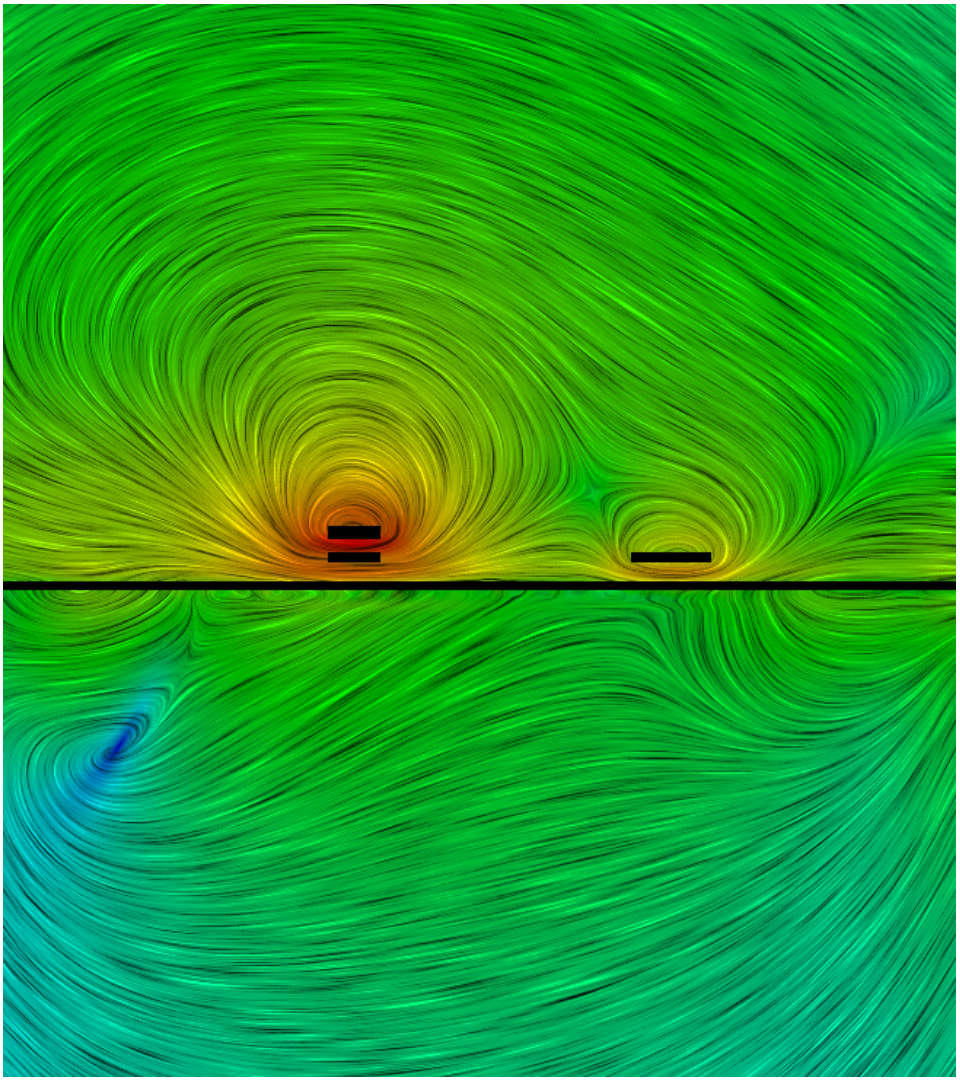


Figure 12.8: Magnetic fields as seen from the cross-section view, for Fig. 12.7

Looking at the GDS layout of the eDFF in Fig. 12.7 the biasing line's return path to ground is placed perpendicular to the mutual coupling line. This causes the return line to have no magnetic coupling with the mutual stucture. The magnetic fields of this topology are shown in Fig. 12.8.

# Chapter 13

# Conclusions

In this thesis a set of basic eSFQ gates where fabricated and the visualization of magnetic fields around the eSFQ circuit layers where calculated using the proposed toolchain. Using MagnetEx the designer can analyse the effect mutual coupling has on a circuit and how the coupling changes due to field changes, the designer can also see the effects of ground plane holes on the surrounding structures of the circuit.

The reported tool can determine the magnetic fields of any superconducting fabrication process. Drastic changes were made to FastHenry to provide the necessary output files that can be read in and processed by the program. The toolkit consists of two stand-alone tools, one for calculating the magnetic fields using rectangular segments (FH engine) and the other using triangular and tetrahedron segments (TH engine). Induced magnetic fields that deteriorate circuit operation, return current in the ground plane and current distributions in circuit layers can now be calculated and visualized in a variety of ways for superconducting circuits. In this thesis we showed examples of magnetic fields and current distribution. Improvements where made in the accuracy and speed of calculating the magnetic fields, which makes it practical to solve and visualize the fields for full circuits.

Future work will focus on implementing support for ferromagnetic materials. Many other techniques was discussed that can be implemented to improve the program depending on the user requirements, such as using the quadtree algorithms explained.

## 13.1 Summary

The goal of this thesis was to setup a basic toolkit that can visualize magnetic fields caused by a variety of sources in superconducting electronic circuit layouts, and to thus give designers the ability to examine circuit layouts for

otherwise hidden weaknesses in magnetic fields. It was important not to just prove the theoretical side of calculating Biot-Savart using FastHenry and InductEx, but to get a system setup correctly for future work. This made it very important to structure the toolkit as simply as possible, therefore the final system is divided into two separate programs. This makes the toolkit dynamic to changes in TH and FFH. Ultimately, it is very difficult to exactly predict what the user wants and therefore future investigation will mainly focus of user experience and feedback.

# Bibliography

[1] S. M. Rao, D. R. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *Antennas and Propagation, IEEE Transactions on*, vol. 30, no. 3, pp. 409–418, 1982.

[2] K. Jackman and C. J. Fourie, "Fast multicore fasthenry and a tetrahedral modeling method for inductance extraction of complex 3d geometries," *in Ext. Abs. 15th Int. Supercon. Elec. Conf. (ISECâ15), Nagoya, Japan, in print*, 2015.

[3] M. Kamon, M. Tsuk, and J. White, "Fasthenry: a multipole-accelerated 3-d inductance extraction program," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 42, no. 9, pp. 1750–1758, Sept 1994.

[4] K. Jackman and C. J. Fourie, "Tetrahedral modelling method for inductance extraction of complex 3D superconducting structures," *EUCAS 2015, Lyon, France*, Sep 2015.

[5] C. J. Fourie, O. Wetzstein, T. Ortlepp, and J. Kunert, "Three-dimensional multi-terminal superconductive integrated circuit inductance extraction," *Superconductor Science and Technology*, vol. 24, no. 12, p. 125015, 2011.

[6] C. Fourie, X. Peng, R. Numaguchi, and N. Yoshikawa, "Inductance and coupling of stacked vias in a multilayer superconductive ic process," *Applied Superconductivity, IEEE Transactions on*, vol. 25, no. 3, pp. 1–4, June 2015.

[7] C. J. Fourie, A. Takahashi, and N. Yoshikawa, "Fast and accurate inductance and coupling calculation for a multi-layer nb process," *Superconductor Science and Technology*, vol. 28, no. 3, p. 035013, 2015. [Online]. Available: http://stacks.iop.org/0953-2048/28/i=3/a=035013

[8] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[9] D. Hearn and M. Baker, *Computer Graphics*. Prentice-Hall International Editions, 1986.

[10] C. Fourie, "Full-gate verification of superconducting integrated circuit layouts with inductex," *Applied Superconductivity, IEEE Transactions on*, vol. 25, no. 1, pp. 1–9, Feb 2015.

[11] D. H. Schaubert, D. R. Wilton, and A. W. Glisson, "A tetrahedral modeling method for electromagnetic scattering by arbitrarily shaped inhomogeneous dielectric bodies," *Antennas and Propagation, IEEE Transactions on*, vol. 32, no. 1, pp. 77–85, 1984.

[12] M. Li and W. C. Chew, "Applying divergence-free condition in solving the volume integral equation," *Progress In Electromagnetics Research*, vol. 57, pp. 311–333, 2006.

[13] S. M. Anton, I. A. B. Sognnaes, J. S. Birenbaum, S. R. OâKelley, C. J. Fourie, and J. Clarke, "Mean square flux noise in squids and qubits: numerical calculations," *Superconductor Science and Technology*, vol. 26, no. 7, p. 075022, 2013.

[14] W. Hafla, A. Buchau, and W. M. Rucker, "Application of fast multipole method to biot-savart law computations," in *Computational Electromagnetics (CEM), 2006 6th International Conference on*. VDE, 2006, pp. 1–2.

[15] C. Weggel and D. Schwartz, "New analytical formulas for calculating magnetic field," *Magnetics, IEEE Transactions on*, vol. 24, no. 2, pp. 1544–1547, 1988.

[16] J. Strain, "Locally corrected multidimensional quadrature rules for singular functions," *SIAM Journal on Scientific Computing*, vol. 16, no. 4, pp. 992–1017, 1995.

[17] P. C. Hammer and A. H. Stroud, "Numerical evaluation of multiple integrals ii," *Mathematical Tables and other Aids to Computation*, pp. 272–280, 1958.

[18] D. Wilton, S. Rao, A. Glisson, D. Schaubert, O. Al-Bundak, and C. Butler, "Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains," *Antennas and Propagation, IEEE Transactions on*, vol. 32, no. 3, pp. 276–281, Mar 1984.

[19] R. Graglia, "Static and dynamic potential integrals for linearly varying source distributions in two- and three-dimensional problems," *Antennas and Propagation, IEEE Transactions on*, vol. 35, no. 6, pp. 662–669, Jun 1987.

[20] G. Aiello, S. Alfonzetti, B. Azzerboni, S. Coco, and G. Tina, "Analytical computation of magnetic vector potential from tetrahedral conductors," *Magnetics, IEEE Transactions on*, vol. 28, no. 5, pp. 2045–2050, 1992.

[21] B. Cabral and L. C. Leedom, "Imaging vector fields using line integral convolution," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '93. New York, NY, USA: ACM, 1993, pp. 263–270.

[22] D. Stalling and H.-C. Hege, "Fast and resolution independent line integral convolution," in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 249–256.

[23] Z. Liu, R. Moorhead, and J. Groner, "An advanced evenly-spaced streamline placement algorithm," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 12, no. 5, pp. 965–972, Sept 2006.

[24] B. Jobard, G. Erlebacher, and M. Hussaini, "Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 8, no. 3, pp. 211–222, Jul 2002.

[25] A. Okada and D. Lane, "Enhanced line integral convolution with flow feature detection," in *In SPIE Vol. 3017 Visual Data Exploration and Analysis IV*, 1997, pp. 206–217.

[26] T. Acharya and P.-S. Tsai, "Computational foundations of image interpolation algorithms," *Ubiquity*, vol. 2007, no. October, pp. 4:1–4:17, Oct. 2007.

[27] P.-S. Tsai and T. Acharya, "Image up-sampling using discrete wavelet transform." in *JCIS*, 2006.

[28] R. Asamwar, K. Bhurchandi, and A. Gandhi, "Interpolation of images using discrete wavelet transform to simulate image resizing as in human vision," *International Journal of Automation and Computing*, vol. 7, no. 1, pp. 9–16, 2010.

[29] E. Dumic, S. Grgic, and M. Grgic, "The use of wavelets in image interpolation: Possibilities and limitations."

[30] W. Hafla, A. Buchau, and W. Rucker, "Efficient computation of source magnetic scalar potential," *Advances in Radio Science*, vol. 4, no. 4, pp. 59–63, 2006.

[31] R. Keys, "Cubic convolution interpolation for digital image processing," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 29, no. 6, pp. 1153–1160, Dec 1981.

[32] J. W. Hwang and H. S. Lee, "Adaptive image interpolation based on local gradient features," *Signal Processing Letters, IEEE*, vol. 11, no. 3, pp. 359–362, March 2004.

[33] S. Yuan, M. Abe, A. Taguchi, and M. Kawamata, "High accuracy bicubic interpolation using image local features," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E90-A, no. 8, pp. 1611–1615, Aug. 2007.

[34] L. Lincoln and R. Gonzalez, "Interpolating leaf quad tree image compression," in *Signal Processing and Communication Systems (ICSPCS), 2013 7th International Conference on*, Dec 2013, pp. 1–7.

[35] A. Tabarraei and N. Sukumar, "Adaptive computations on conforming quadtree meshes," *Finite Elem. Anal. Des.*, vol. 41, no. 7-8, pp. 686–702, Apr. 2005.

[36] A. Scholefield and P. Dragotti, "Quadtree structured image approximation for denoising and interpolation," *Image Processing, IEEE Transactions on*, vol. 23, no. 3, pp. 1226–1239, March 2014.

[37] Z. Wang, "A quadtree-based adaptive cartesian/quad grid flow solver for navier-stokes equations," *Computers  Fluids*, vol. 27, no. 4, pp. 529 – 549, 1998.

[38] P. J. Frey and L. MARECHAL, "Fast adaptive quadtree mesh generation," in *in: Proceedings of the Seventh International Meshing Roundtable*, 1998, pp. 211–224.

[39] A. Johnson. (1999) Clipper - an open source freeware library for clipping and offsetting lines and polygons. [Online]. Available: http://www.angusj.com/delphi/clipper.php

[40] B. Kim, P. Tsiotras, J.-M. Hong, and O.-Y. Song, "Interpolation and parallel adjustment of center-sampled trees with new balancing constraints," *Vis. Comput.*, vol. 31, no. 10, pp. 1351–1363, Oct. 2015.

[41] C. Gaspar, "Flow modelling using quadtrees and multigrid technique."

[42] K. Lindsay and R. Krasny, "A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow," *Journal of Computational Physics*, vol. 172, no. 2, pp. 879–907, 2001.

[43] ——, "A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow," *Journal of Computational Physics*, vol. 172, no. 2, pp. 879–907, 2001.

[44] Y. Yamanashi, "Investigation of influence of trapped flux in moats on superconducting circuit operation," private communication, 2015.

[45] A. Golubov and U. Hartmann, "Electronic structure of the abrikosov vortex core in arbitrary magnetic fields," *Physical review letters*, vol. 72, no. 22, p. 3602, 1994.

[46] B. Ebert, T. Ortlepp, and F. Uhlmann, "Experimental Study of the Effect of FLuc Trapping on the Operation of RSFQ Circuits," *IEEE Transactions on Applied Superconductivity*, vol. 19, no. 3, pp. 607–610, Jun 2009.

[47] M. Jeffery, "Magnetic imaging of moat-guarded superconducting electronic circuits," *Applied Physics Letters*, vol. 67, no. 12, pp. 1769–1771, Sep 1995.

[48] M. Volkmann, A. Sahu, C. Fourie, and O. Mukhanov, "Experimental investigation of energy-efficient digital circuits based on esfq logic," *Applied Superconductivity, IEEE Transactions on*, vol. 23, no. 3, pp. 1 301 505–1 301 505, June 2013.

[49] Y. Yamanashi, T. Nishigai, and N. Yoshikawa, "Study of LR-Loading Technique for Low-Power Single Flux Quantum Circuits," *IEEE Transactions on Applied Superconductivity*, vol. 17, no. 2, pp. 150–153, Jun 2007.

[50] T. Ortlepp, O. Wetzstein, S. Engert, J. Kunert, and H. Toepfer, "Reduced Power Consumption in Superconducting Electronics," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. pp. 770–775, Jun 2011.

[51] S. Polonsky, "Delay insensitive RSFQ circuits with zero static power dissipation," *IEEE Transactions on Applied Superconductivity*, vol. 9, no. 2, pp. pp. 3535–3538, Jun 1999.

[52] D. Kirichenko, S. Sarwana, and A. Kirichenko, "Zero Static Power Dissipation Biasing of RSFQ Circuits," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, Jun 2011.

[53] M. Suzuki, M. Maezawa, and F. Hirayama, "Effects of magnetic fields induced by bias currents on operation of RSFQ circuits," *Physica C*, pp. 412–414, Jun 2004.

[54] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, "An adiabatic quantum flux parametron as an ultra-low-power logic device," *Supercond. Sci. Technol*, pp. pp. 1–5, Jan 2013.

[55] O. Mukhanov, "Energy-Efficient Single Flux Quantum Technology," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, Jun 2011.

[56] I. V. Vernik, S. B. Kaplan, M. H. Volkmann, A. V. Dotsenko, C. J. Fourie, and O. A. Mukhanov, "Design and test of asynchronous esfq circuits," *Superconductor Science and Technology*, vol. 27, no. 4, p. 044030, 2014.

[57] P. Bunyk, K. Likharev, and D. Zinoviev, "RSFQ Technology: Physics and Devices," *International Journal of high Speed Electronics and Systems*, vol. 11, no. 1, pp. pp. 257–305, 2001.

[58] K. Likharev and V. Semenov, "RSFQ Logic/Memory Family: A New Josephson-Junction Technology fot Sub-Terahertz-Clock-Frequency Digital System," *IEEE Transactions on Applied Superconductivit*, vol. 1, no. 1, Mar 1991.

[59] N. Yoshikawa and Y. Kato, "Reduction of power consumption of RSFQ circuits by inductance-load biasing," *Supercond. Sci. Technol.*, pp. pp. 918–920, Jun 1999.

# Appendices

# Code Implementation

## Main Function

The **sys** variable used in the program was a method learned by analyzing the code of the original FastHenry program.

```
int main(void)
{
   SYS *sys = (SYS *)calloc(1, sizeof(SYS));

   Paths sol;

   init_var(sys);

   read_fil_len(sys->fil_file, sys);
   read_mat(sys->mat_file, sys);
   read_fil(sys->fil_file, sys);

   side_view(sys);
   create_grid(sys);
   seg_to_array(sys);
   seg_to_poly(sol, sys);

   biot_savart(sys);
   bicubic(sys);
   lic(sys);
   write_bmp(sol, sys);

   return 0;
}
```

The variable refers to a structure which contains pointers to many of the structures and arrays needed by most functions. The **main** function is divided into six parts:

- Initializing many of the variables and creating data structure using the **sys** variable.

- Reading in files, creating arrays and saving segments of the meshed structures in the intended data structures.
- Creating the observation grid and geometric processing.
- Calculating Bio-Savart and using bicubic interpolation.
- Implementing the LIC algorithm, applying visualization filters and enhancement methods.
- Draw image and create magnitude legend.

**cross_section:** Determine which segments are sliced and saves them in an array of segment data structures.

**create_grid:** Uses the array of segments to calculate the dimensions of the observation grid. Two arrays are created, one containing the points that will map to each image pixel and the other that will represent the points at which Biot-Savart will be applied.

**seg_to_array and seg_to_poly:** Uses the Clippers library to generate data structures that contains the vertices and edges of the sliced segments. The generated observation grid are then filled with ones and zeros, to represent when a grid point is inside or outside a superconducting layer.

**biot_savart, bicubic, lic, and write_bmp:** These functions are self explanatory with some notice points:

- **biot_savart**: Can manually choose to include or exclude points inside the layers.
- **bicubic**: Can choose between the different bicubic enhancement algorithms implemented and explained.
- **lic**: Can change filters, where each filter is defined by its own function. Can iterate between using the Double LIC method or not, and also toggle Fast LIC of or on.

These changes can only be made by the developer and not by the user. Many of these dynamic functions are only for debugging or experimental purposes. The default program comes with a set of default settings. If the user wants to make changes, then the InductEx team will have to be consulted. This also gives us a tracking system to which functionality the user wants.

# RSFQ Basics

One of the main problems with Josephson junction electronics is how to pass information of the current flux state through a system. A junction that switches can not send a flux pulse on to infinity, since the natural inductance of a superconducting wire causes a phase drop of [57]

$$\Delta\varphi = (2\pi/\Phi_0)LI \tag{.0.1}$$

as explained in Section 13.1. This phase drop will ultimately lead to phase equilibrium between two nodes, which in turn will result in no current flow and thus no flow of information. In Rapid Single Flux Quantum logic the phase is amplified by repeatedly switching junctions as the pulse propagates through a circuit. The circuit resonsible for this pulse transfer is known as the Josephson Transmission Line (JTL) [58].

RSFQ devices can be classified into twee subsections:

- *asynchronous:* Circuits that does not save a flux pulse in a SQUID and generates an output pulse on the arrival of an input pulse
- *synchronous:* Circuits that saves internally a flux and only generates an output upon arrival of a clock pulse.

Understanding the workings of the most basic RSFQ cell, the JTL, is enough to grasp the basic concepts of RSFQ logic. In the initial state each junction is biased with approximately

$$I_1 = 0.7I_C, \tag{.0.2}$$

where $I_C$ is the critical current of the biased junction. A dc phase drop across the junction occurs because of the biasing current, equal to

$$\phi_1 = \arcsin(I_1/I_C) \approx \pi/3 \tag{.0.3}$$

Junction $J_1$ will switch first, this will cause a $2\pi$ phase increase at node $A$. The phase increase happens due to the physical *level-up* of the quantum numbers of the quasiparticles affected by the junction switch.

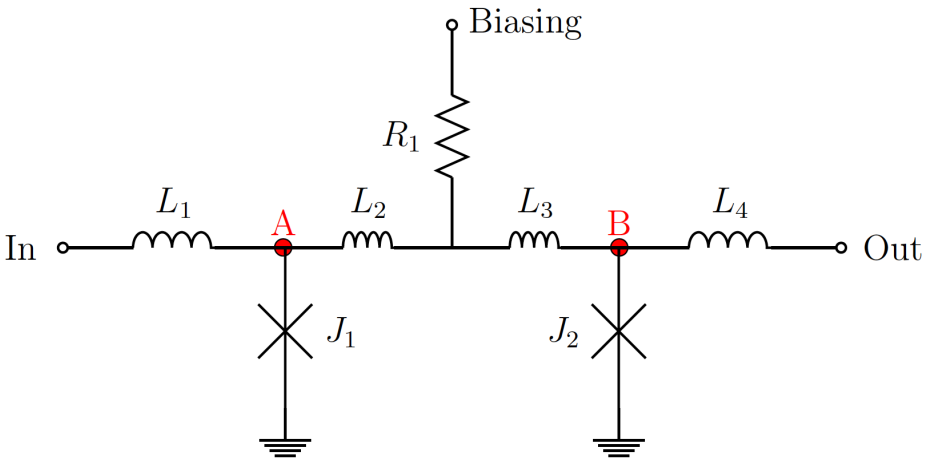



Figure 1: Basic JTL

Quasiparticles at node $A$ that have a higher quantum number are described by a wavefunction with a phase of approximately $\pi/3 + 2\pi$. To equalize the phase difference in the system, current flows from node $A$ to node $B$. Switching of $J_1$ increases the current in the loop $J_1 - L_2 - L_3 - J_2$ by

$$\Delta I = \Phi_0/(L_{J1} + L_2 + L_3 + L_{J2}) \tag{.0.4}$$

where $L_{J1}$ is the non-linear inductance of the junction. As the flux pulse (which is a voltage pulse) propagates though the superconductor it is converted to current due to the natural inductance characteristics of the superconducting wire. As explained this will cause a phase drop, but just before the phase drops to much, the next junction switches causing a phase amplification of $2\pi$. The current going through an inductor during the passage of a SFQ pulse is given by [59]

$$I = \frac{\Phi_0}{L_b} \tag{.0.5}$$

From equation (.0.1) if the inductor $L_b$ between the two junctions are too large then

$$\Delta\varphi < (2\pi/\Phi_0)LI \tag{.0.6}$$

which will result in the second junction, $J_2$ not switching. A junction that switched can only increase its phase by $2\pi$, because

$$\phi = 2\pi\frac{\Phi}{\Phi_0} \tag{.0.7}$$

Even if $J_2$ were to switch and increase the phase of node $B$ by $2\pi$, there will still be a phase difference between node $A$ and node $B$, since the phase dropped by more than $2\pi$ before the pulse arrived at $J_2$. Thus, $J_2$ will not switch and the pulse will be stored in the loop $J_1 - L_2 - L_3 - J_2$.

# Junction Constants

In the design of SFQ electronics the most important aspect is to know the different functionalities of the Josephson junction. In this section a brief introduction will be given to the basic timing constants of the Josephson junction. Most of these can be found in any Josephson junction text book, but the most important parameters are given in this section for the reader to get a basic graps of this device. First of all, it is important to know that a Josephson junction can be represented by a nonlinear inductor

$$L_J = \frac{\Phi_0}{2\pi I_c}. \tag{.0.8}$$

Then we define the following linear time constants

$$\begin{aligned} \tau_1 &= RC \\ \tau_2 &= \frac{\Phi_0}{2\pi I_c R} \\ \tau_0 &= \sqrt{\tau_1 \tau_2}. \end{aligned}$$

where $R$ is the intrinsic resistance represented by the collisions made by the Cooper-pairs. Josephson junctions can be designed to be either overdamped, underdamped or critically damped. The parameter that determines this is the $\beta_c$ parameter.

$$\beta_c = \frac{\tau_1}{\tau_2} = \frac{R^2 C}{L_J}. \tag{.0.9}$$

These equations state that the junction switching speed is limited by either $\tau_1$ or $\tau_2$. If $\beta_c > 1$, the determinant time constant is $\tau_1$. If $\beta_c < 1$, the determinant time constant is $\tau_2$. If $\beta_c = 1$, then both constants care equal weight. The time it takes for a junction to return to its superconduting state after it has switched is given by

$$\Delta t = \frac{\Phi_0}{2 I_c R} \tag{.0.10}$$

# Quantum Physics

Most papers and thesises about Josephson junction electronics describes the basic workings of the Josephson junction from a very basic point of view. Most of these explanantions start by using the basic junction equations without explaining its origins. The focus of this chapter falls on the quantum mechanical explanation of the Josephson junction.

## Critical Current and SFQ Pulse Detection

Particle movement in a superconductor is disturbed by collisions with other particles in the superconductor. To an approximation these particles behaves the same way as if there were no interactions with other particles at all, except with a different mass. These particles with a different mass are called quasiparticles. Quasiparticle theory is a simplified method to the *quantum mechanical many-body problem.*

In semiconducting the many-body problem can become very complex, since the wavefunction of the system becomes impractical. In these cases a set of approximations needs to be made, such as *perturbation theory* and *Green's function-based* methods. Quasiparticles plays an important role in Josephson junction electronics, because the system goes into the resistive state (becomes semiconducting) for a couple of picoseconds each time a junction switches.

The approximation of why an electron has a different mass in a conductor can very briefly be described. If an electron were to be send from point $A$ to point $B$ by means of a force, with energy $E$ and if this electron where to reach the speed of light and still receive energy $E$ from this source (biasing current source). Then according to Einstein's equation

$$E = mc^2 \tag{.0.11}$$

an increase in energy on the left of the equation, must correspond to an increase on the right. Since the speed of light, $c$, is a constant the only thing that can increase is the mass of the particle. Looking at a system with no particle collisions the mass of the electron will be close to that of its original

mass. But in a system with complex particle movement, more energy will have to be used to send the electron through the conductor. This corresponds to an increase in the electron mass. It is this sudden change in mass of the particles when a junction switches that produces a voltage pulse.

Once a superconducting material is cooled to a certain critical temperature, the quasiparticles groups together in pairs to form Cooper-pair particles. Cooper-pairs form a Bose-Einstein condensate which is a state of matter where the bosons occupy the lowest quantum state and has a mass of twice that of normal quasiparticles. In the lowest quantum state certain quantum effects becomes measurable, which forms the basis of Josephson junction electronics. The large mass and uniformity of the Cooper-pairs are what makes them a macroscopic quantum phenomenon.

The basic function of a Josephson junction is to measure a change in this *macroscopic quantum* state. Each time a junction switches the Cooper-pairs jumps to the next quantum state or energy level, becomes semiconducting for a couple of picseconds and then returns to its superrconducting state. In other words their quantum numbers increases and this causes an increase in the phase of the quantum wavefunction, which in turn causes a voltage pulse to be generated. When the Cooper-pairs break, the system loses mass. This mass is turned into energy, according to $E = mc^2$, that can be observed as a voltage pulse. The following equation gives the critical current $I_c$ of a junction, which is the minimum current that needs to go through a junction to make it switch

$$I_c R_n = \frac{\pi \Delta}{2e} \tag{.0.12}$$

Here $R_n$ is the intrinsic quasiparticle resistance, and $2\Delta$ is the superconducting energy gap. When the current through a junction exceeds this critical current, the energy through the material of the junction is more than the energy gap, $2\Delta$, of the Cooper-pairs. This causes the Cooper-pairs to jump to a higher energy state or quantum state, which releases a single flux pulse (SFQ).

## Junction Voltage and Current

Consider two systems where the potential energy between them are proportional to the phase difference between them. If we have two entangled particles with the difference between there potential energies, $\Delta U = U_1 - U_2$. Then the difference between the phases of there wavefuntions, $\varphi = \varphi_1 - \varphi_2$ evolves in time as:

$$\frac{d\varphi}{dt} = -\frac{\Delta U}{\hbar} \tag{.0.13}$$

At very low temperatures two electrons bound and form a *Cooper-pair*. One electron with a *spin up* and the other with a *spin down*. These particles become

strongly entangled and both the momentum and spin of this Cooper-pair adds up to zero and thus behaves as a *Bose particle*. The effective potential energy of the Cooper-pair is then $U = -2e\phi$, where $\phi$ is the electrostatic potential. With $V = \phi_1 - \phi_2$ equation (.0.13) becomes

$$\frac{d\varphi}{dt} = \frac{2e}{\hbar}V \tag{.0.14}$$

Though this equation was only derived from the differences between two Cooper-pair particles the same equation can be used to describe the effect between two superconducting materials, since each materials can be described by a single wavefunction. Equation (.0.32) states that the phase difference between two superconducting materials can be changed by changing the *voltage* between them. This is done by separating two superconducting materials with a *Josephson junction*.

The canonical momentum is the kinetic momentum that changes due to the magnetic field, and therefore in quantum mechanics the canonical momentum rather than the kinematic momentum is used, which is also given by

$$\mathbf{P} = -i\hbar\nabla \tag{.0.15}$$

Placing this into the Hamiltonian of a particle inside a magnetic field we get

$$H = \frac{p^2}{2m} + q\phi = -\frac{\hbar^2}{2m}(\nabla - \frac{iq}{\hbar}\mathbf{A})^2 + q\phi \tag{.0.16}$$

Then the Schrodinger equation becomes

$$-\frac{\hbar^2}{2m}(\nabla - \frac{iq}{\hbar}\mathbf{A})^2\psi + q\phi\psi = E\psi \tag{.0.17}$$

Then the *probability current density* can be calculated:

$$\begin{aligned}\mathbf{j} &= \frac{\hbar^2}{2im}(\psi^*(\nabla - \frac{iq}{\hbar}\mathbf{A})\psi - \psi(\nabla - \frac{iq}{\hbar}\mathbf{A})\psi^*) \\ &= \frac{1}{2m}(\psi^*\mathbf{P}\psi - \psi\mathbf{P}\psi^*) \\ &= \frac{\hbar}{m}|\psi|^2(\nabla\varphi - \frac{q}{\hbar}\mathbf{A}).\end{aligned} \tag{.0.18}$$

From this the current can be derived as

$$I(x,t) = \int j_x \mathrm{d}y\mathrm{d}z = \frac{\hbar}{m}|\Psi(x,t)|^2\frac{\partial\varphi}{\partial x}. \tag{.0.19}$$

which shows a direct correspondence between the phase and the current in a superconducting circuit. From observation, Josephson, showed that the current changes sinusoidally with respect to the phase:

$$I = I_c \sin\varphi \tag{.0.20}$$

where $I_c$ is the *critical current*. If the applied voltage stays constant, then the current oscillates in time with the following *angular frequancy*:

$$w_c = \frac{2e}{\hbar} V_{constant} \tag{.0.21}$$

# Phase Change in a Circuit

An important physical equation that can be used in SFQ circuit design will be derived in this section without using too much mathematical tools. Focus will fall on the basic logical thought procedures followed to come to sensible mathematical conclusions. Let us look at the Double-Slit experiment shown in the figure below.

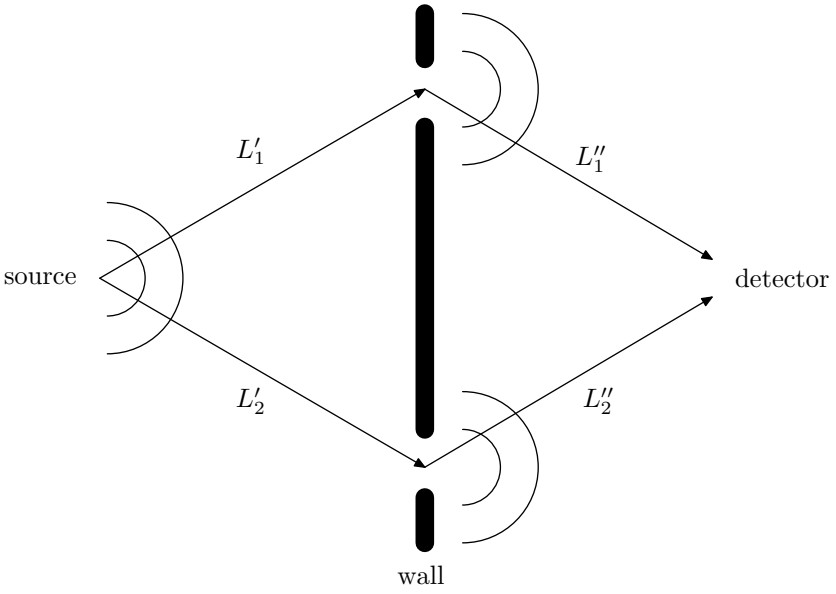



Figure 2: Double-Slit Experiment

The probability of obtaining a particle at the particle detector on the right, is $w(\mathbf{r}, t) = \Psi(\mathbf{r}, t)\Psi^*(\mathbf{r}, t)$. If the traveling wave has an energy $E$, then its wavefuntion can be described by

$$\Psi(\mathbf{r}, t) = \psi(\mathbf{r}) e^{-iEt/\hbar}, \tag{.0.22}$$

where $\psi(\mathbf{r})$ is the amplitude and the exponential term shows that the wave vanishes with time. This does not mean the particle vanishes with time, but only that the probability of finding the particle in space-time is bounded. The difference of the total wave phase accumulations along each of the two alternative trajectories are given by

$$\varphi_{12} = k(L_2 - L_1) = \oint_C \mathbf{k} \cdot \mathrm{d}\mathbf{r} \tag{.0.23}$$

Where $L_2 = L_2' + L_2''$ and $L_1 = L_1' + L_1''$. This very basic equation states that the phase of a supercoducting current changes as the current propagates through the superconductor. From this a general equation can be derived that is directly proportional to the product of the inductor and current passing through the currently defined superconducting wire, multiplied by a specific constant that will consist of the flux constant $\Phi_0$. The logical reasoning behind this explanation is the fact that the path the SFQ pulse in a circuit follows are only defined by the current is represents and the inductance it passes through, then like in any other physical equation a constant has to be introduced. Finally, this equation can be formulated as

$$\Delta\varphi = (2\pi/\Phi_0)LI \tag{.0.24}$$

A direct mathematical derivation of this equation can be found in [57], but explaining this equation using the most basics of quantum physics improves once understanding of how to handle and munipulate SFQ pulses. A SFQ pulse only has a certian propability of reaching its intended target. A method to increase this propability in circuit design is to repeatedly switch junctions, to reset the phase and thus increase the propability again. This logic gate is called a Josephson Transmission Line and is explained in Chapter 13.1.

# Quantization

This section explains why there is no magnetic field and thus no current inside a superconductor. Also the reason for flux quantization is explained. Let us look at the effect of a magnetic field on quantum interference. To do this we change the Double-Slit experiment as shown in Figure 3. The magnetic field

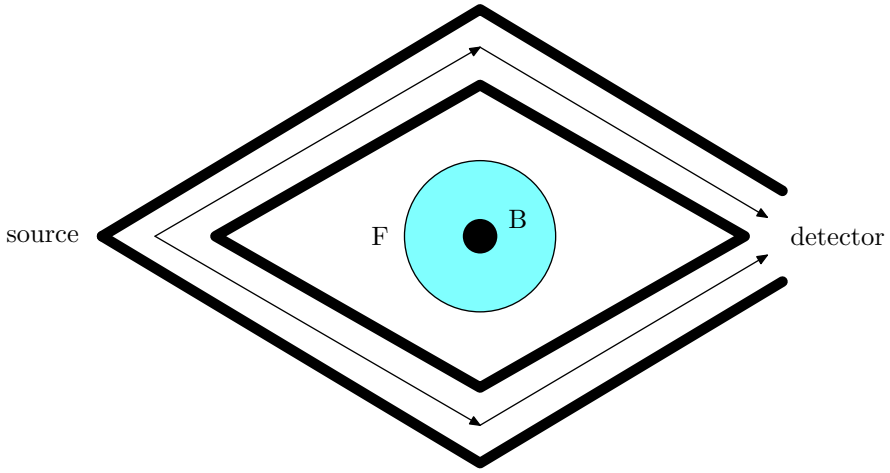



Figure 3: Double-Slit Experiment with **B**-field and Bounded Path

**B** does not make contact with the wavefuntion of the particles. In classical

mechanics there will be no magnetic effect on the particle, but in quantum mechanics there is. From Maxwell's equations the electric field $\mathbf{E}(t,x)$ and the magnetic field $\mathbf{B}(t,x)$ can be represented in terms of a scalar potential $A_0(t,x)$ and a vector potential $\mathbf{A}(t,x)$. Gauge invariance means that a whole class of scalar and vector potentials can be described by the same electric and magnetic potentials. These different potentials are however related by *gauge transformations*. Maxwell's equations can be proven to be gauge invarient, but the canonical momentum is not gauge invarient. Since the current was derived from the canonical momentum and the Hamiltonian, the current in a superconductor is also not gauge invariant. Which means that we can measure the current directly, but not the phase, since the phase is not a real quantity.

The phase difference may be affected by the magnetic field, even if the magnetic field vectors never interacts with the particles. This means that the current density in the center of the superconductor will always be the same whether a magnetic field is penetrating the superconductor or not. Most of the time the current density in the center of the superconductor will be zero, but when a SFQ pulse is generated the current density changes. When this happens the following relation must still be valid

$$\mathbf{j}(\mathbf{r})|_{B\neq 0} = \mathbf{j}(\mathbf{r})|_{B=0} \tag{.0.25}$$

then from equation (.0.18)

$$\begin{aligned} \nabla\varphi(\mathbf{r})|_{B\neq 0} - \frac{q}{\hbar}\mathbf{A} &= \nabla\varphi(\mathbf{r})|_{B=0} \\ \nabla\varphi(\mathbf{r})|_{B\neq 0} &= \nabla\varphi(\mathbf{r})|_{B=0} + \frac{q}{\hbar}\mathbf{A}. \end{aligned} \tag{.0.26}$$

After integrating

$$\varphi_{12}|_{B\neq 0} = \varphi_{12}|_{B=0} + \frac{q}{\hbar}\oint_C \mathbf{A}\cdot \mathrm{d}\mathbf{r}, \tag{.0.27}$$

here $C$ is the closed loop formed by the two paths of the traveling waves. The magnetic field flux is

$$\Phi = \int B_n \mathrm{d}^2 r, \tag{.0.28}$$

equation (.0.27) can be rewritten as

$$\varphi_{12}|_{B\neq 0} = \varphi_{12}|_{B=0} + \frac{q}{\hbar}\Phi \tag{.0.29}$$

This means a shift in the fringes of the interference pattern, is proportional to the magnetic flux and is known as the *Aharonov-Bohm* effect. The *normal magnetic flux quantum*

$$\Phi_0 = \frac{2\pi\hbar}{2e} = \frac{h}{2e} = 2.07\times 10^{-7}\ Wb \tag{.0.30}$$

is the flux necessary to change the phase by $2\pi$, which causes the interference pattern to shift by one period.

$$\varphi_{12}|_{B\neq 0} = \varphi_{12}|_{B=0} \pm 2\pi \frac{\Phi}{\Phi_0} \tag{.0.31}$$

A change of the magnetic flux in time causes a vortex-like electric field $\Delta\varepsilon$ around the magnetic field. This electric field is not restricted to the magnetic field's location. The field's magnitude can be found by integrating equation $\varepsilon = -\nabla\phi - \frac{\partial \mathbf{A}}{\partial t}$,

$$\Delta V = \oint_C \Delta\varepsilon \cdot \mathrm{d}\mathbf{r} = -\frac{d\Phi}{dt} \tag{.0.32}$$

This equation stats that a change in magnetic flux in the superconductor causes a voltage change. Mathematically it can be shown that if the magnetic flux changes, then the spatial parts, $\psi_j$, of the wavefunction does not change. The potential energy difference $\Delta U = q\Delta V$ between the two points does change. Looking at the *Josephson effect* (.0.32) we find

$$\frac{d\varphi_{12}}{dt} = -\frac{\Delta U}{\hbar} = -\frac{q}{\hbar}\Delta V = \frac{q}{\hbar}\frac{d\Phi}{dt} \tag{.0.33}$$

In a superconductor the current and magnetic flux only penetrates the material till a certain depth, the London penetration depth. From equation (.0.18) this means the following equation has to be true in the center of the superconducting loop

$$\nabla\varphi_{12} - \frac{q}{\hbar}\mathbf{A} = 0. \tag{.0.34}$$

The current traveling in this superconducting loop is due to the flow of Cooper-pair particles. Explained in Section 13.1, the large mass of the Cooper-pair particles increases their potential energy and therefore they can travel across the Josephson junction without disturbing the actual wavefunction properties, a phenomenon known as *quantum tunneling*.

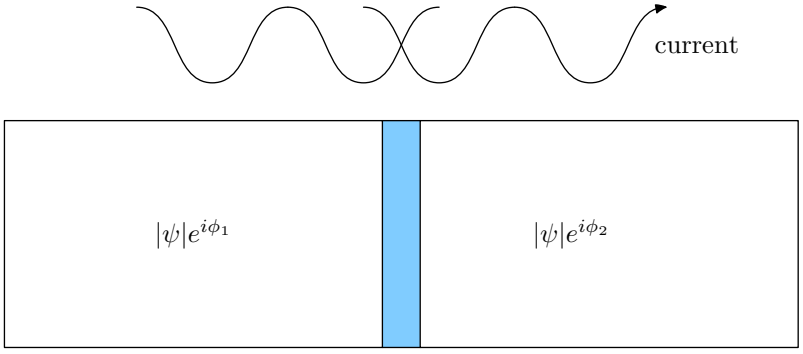



Figure 4: Junction with current Out of Phase

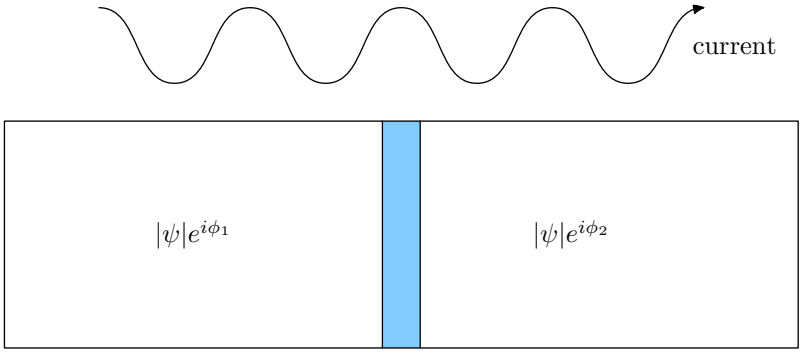



Figure 5: Junction with current In Phase

The wavefunction crossing the Josephson junction must be continious and therefore the phase difference between the two superconductors separated by the junction can only differ by integer multiples of $\varphi_{12} = 2\pi n$.. If the phase between the two materials at a certain point differ by any other mount, the wavefunction is broken and at that point the material losses its superconductivity, until the phase difference is recovered.

Figure 4 shows a discontinious wavefunction and Figure 5 shows the correct wavefunction. We know

$$\Phi = \oint_C \mathbf{A} \cdot \mathrm{d}\mathbf{r}. \tag{.0.35}$$

Integrating (.0.34),

$$\varphi_{12} = \frac{q}{\hbar} \oint_C \mathbf{A} \cdot \mathrm{d}\mathbf{r}. \tag{.0.36}$$

Then substituting (.0.35) into (.0.36), using (.0.30) and since $q = 2e$ we prove our discussion

$$\begin{aligned} \Phi &= \frac{\hbar}{q}\varphi_{12} \\ &= \frac{h}{2\pi q}(2\pi n) \\ &= \frac{h}{2e}n \\ &= n\Phi_0. \end{aligned} \tag{.0.37}$$

This tells us that in a superconducting material the magnetic flux only exists in packages of integer multiples of the defined constant $\Phi_0$. This phenomenon is called the *flux quantization effect*.